Parallelizing Ant Colony Optimization via Area of Expertise Learning

THESIS

Adrian A. de Freitas, Second Lieutenant, USAF

AFIT/GCS/ENG/07-15

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GCS/ENG/07-15

Parallelizing Ant Colony Optimization via Area of Expertise Learning

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Adrian A. de Freitas, B.S.C.S.

Second Lieutenant, USAF

September 2007

AFIT/GCS/ENG/07-15

Parallelizing Ant Colony Optimization via Area of Expertise Learning

Adrian A. de Freitas, B.S.C.S.
Second Lieutenant, USAF

Approved:

Maj Christopher B. Mayer, PhD (Chairman)	date
Dr. Gilbert L. Peterson (Member)	date
Dr. Gary B. Lamont (Member)	date

AFIT/GCS/ENG/07-15

Abstract

Ant colony optimization algorithms have long been touted as providing an effective and efficient means of generating high quality solutions to NP-hard optimization problems. Unfortunately, while the structure of the algorithm is easy to parallelize, the nature and amount of communication required for parallel execution has meant that parallel implementations developed suffer from decreased solution quality, slower runtime performance, or both. This thesis explores a new strategy for ant colony parallelization that involves Area of Expertise (AOE) learning. The AOE concept is based on the idea that individual agents tend to gain knowledge of different areas of the search space when left to their own devices. After developing a sense of their own expertness on a portion of the problem domain, agents share information and incorporate knowledge from other agents without having to experience it first-hand. This thesis shows that when incorporated within parallel ACO and applied to multi-objective environments such as a gridworld, the use of AOE learning can be an effective and efficient means of coordinating the efforts of multiple ant colony agents working in tandem, resulting in increased performance.

Based on the success of the AOE/ACO combination in gridworld, a similar configuration is applied to the single objective traveling salesman problem. Yet while it was hoped that AOE learning would allow for a fast and beneficial sharing of knowledge between colonies, this goal was not achieved, despite the efforts detailed within. The reason for this lack of performance is due to the nature of the TSP, whose single objective landscape discourages colonies from learning unique portions of the search space. Without this specialization, AOE was found to make parallel ACO faster than the use of a single large colony but less efficient than multiple independent colonies.

Acknowledgements

I would like to thank Major Mayer for putting up with me and making sure that I stayed on track, and my family for constantly encouraging me to try my best. Finally, I would like to thank my pets for reminding me that when you don't have anything else to say, a cat on the keyboard can at least get you a few more pages.

Adrian A. de Freitas

Table of Contents

Page

List of Figures

List of Tables

Parallelizing Ant Colony Optimization via Area of Expertise Learning

I. Introduction

In many ways, combinatorial optimization problems are the Achilles' heel of modern-day computers. Although they are intuitively simple in concept, the sheer dimensionality of these problems prevent brute force approaches from generating solutions for all but the most trivial instances. Ant colony optimization (ACO) is a simple metaheuristic that can effectively solve problems in these domains. Unfortunately, while the technique is capable of solving optimization problems that are too large for complete enumeration methods to handle, ACO is also prone to slowdown as the size of the problem increases. In order to speed up ACO, many research attempts have focused on parallelizing the algorithm. This thesis looks at a novel parallelization technique for ACO in which multiple colonies work on the same problem, specialize in different aspects of it, and then collaborate with one another in order to develop a more thorough understanding of the search space. This "area of expertise" technique is applied to two problem domains: gridworld and the traveling salesman problem.

1.1 Motivation

ACO is a metaheuristic that generates high quality solutions to many types of NP-hard problems ranging from the traveling salesman problem to the multidimensional knapsack problem. First proposed by Dorigo and Gambardella [16], ant colony algorithms get their name from their ability to approximate the swarm-like behavior of their real-life counterparts. By themselves, ants are relatively simple agents that are capable of performing extremely limited tasks. As part of a larger colony, however, the primitive actions of multiple ants combine to form impressively complex behavior. Because ants in an ACO colony learn from each other as they explore the

search space, this approach generates solutions that are either equivalent or close to those found by a complete enumeration approach. More importantly, though, is the fact that these near-optimal solutions can be found in a fraction of the time.

Given its focus on multiple, independent ant agents, an obvious extension of the ant colony framework is to implement the algorithm in a parallel environment. One of the main attractions of such an approach is that it allows for the concurrent sharing of knowledge between multiple colonies working in tandem. This in theory should allow each colony to learn more about the search space than would be possible otherwise. Unfortunately, while the parallelization of ACO has been the subject of intense study [6], the results of recent research efforts indicate that little progress has been made. Although the structure of ant colony algorithms easily facilitates a division of labor between various processing units, there has yet to be a means of sharing, or *synchronizing* knowledge between multiple colonies in a manner that is beneficial to all. In fact, up to this writing, every implementation of cooperative learning within ACO has resulted in either increased computational complexity and decreased solution quality, or both. Thus, until significant progress is made, the ant colony framework is relegated to a single processor environment.

Considering the difficulties encountered thus far in parallelizing ACO, it is clear that a fundamental rethinking of the synchronization step is necessary. The purpose of this thesis is to examine a novel approach to knowledge sharing known as Area of Expertise (AOE) learning [1]. Unlike other cooperative learning techniques, AOE focuses on identifying areas of the search space where each agent possess expert knowledge. By sharing these expert regions with one another, each colony is then able to develop a more comprehensive understanding of the search space than it could on its own. Although AOE learning has been successfully utilized in the field of robotics [1], the approach has never been combined into the ACO framework. Thus, this thesis presents the first step in what could prove to be an exciting new direction for ant colony research, as it examines whether or not collaboration between multiple colonies working in tandem overcome the inherent problems of parallelizing ACO.

1.2 Research Objectives

The overall objectives to be accomplished in this thesis are 1) to develop a means of integrating AOE learning within the parallel ACO framework 2) to demonstrate the proof-of-concept of AOE learning in parallel ACO using the gridworld domain, and 3) to study the effects of AOE learning in parallel ACO when applied to the traveling salesman problem.

1.3 Main Contributions

The main contributions this thesis are:

1. The creation of a new methodology that incorporates AOE learning within parallel ACO;
2. Identifying that the effectiveness of AOE learning largely varies depending upon when agents share expert knowledge with one another. An analysis of this technique on gridworld revealed that the best time to share is just before agents' AOEs overlap. When examined within the context of the TSP, however, failed to reveal a similar set of readily identifiable conditions;
3. The discovery that the combination of AOE learning and parallel ACO in multi-objective environments leads to increased runtime performance and solution quality.

1.4 Thesis Outline

The following chapters of the thesis are organized as follows. Chapter II provides background information including a formulation of the TSP and gridworld problems, a survey of recent attempts at parallelizing ACO, and a description of the AOE learning strategy. This is followed by a discussion of the methodology used to incorporate AOE learning within ACO in Chapter III. Chapter IV provides an analysis of the impact of AOE learning when applied to a multi-objective gridworld environment as well as the single objective traveling salesman problem. Finally, Chapter V concludes with

a discussion of all pertinent information obtained from this study, as well as grounds for future research.

II. Background

This chapter provides an overview of related work pertaining to the effectiveness of parallel ant colony algorithms at solving NP-hard optimization problems. It begins with a discussion of the traveling salesman and gridworld problem domains, both of which are well known NP-hard problems and easily mapped to the ant colony framework. Following this is a description of the ant colony optimization technique and the results of various attempts at parallelizing the algorithm. Finally, the area of expertise strategy is introduced as a means of facilitating cooperative learning between multiple ant colonies running in parallel.

2.1 Traveling Salesman Problem

The traveling salesman problem (TSP) is often considered to be the de facto NP-hard optimization problem. First formulated in the 1930s, TSP has since been studied in great detail [21] and was one of the first problem domains successfully mapped to the ant colony framework [15]. Figure 2.1 provides a visual representation of a sample TSP instance.

The TSP is naturally expressed as a weighted graph structure. Nodes in the graph correspond to locations (i.e. cities) while edges denote a path from one node to the other. A cost, oftentimes equated to distance, is associated with each edge. For the purposes of simplicity, Figure 2.1 only illustrates paths from each city to its nearest neighbors. Typically, though, it is assumed that the graph is fully connected

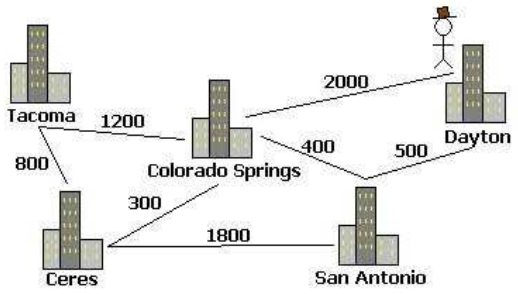

Figure 2.1: A Sample Traveling Salesman Problem.

and non-directed, meaning that it is possible to travel from one node to any other and vice versa.

Given a graph similar to the one described above, the TSP can best be described as finding the cheapest round-trip route, or *Hamiltonian Circuit,* which visits each node exactly once [2]. In order to provide a more formal description of this problem statement, let e_{ij} be an edge that connects node i to j. Also, let a cost, c_{ij}, be associated with traveling from node i to j along e_{ij}. If a Hamiltonian Circuit, S, consists of a set of edges, then the objective of the TSP is to satisfy the following criteria:

$$MIN\ S : \sum_{e_{ij} \in S} c_{ij} \tag{2.1}$$

Looking back at Figure 2.1, the circuit {Tacoma, Colorado Springs, Dayton, San Antonio, Ceres, Tacoma} is the Hamiltonian Circuit that provides a minimal cost.

Although the TSP is intuitively simple from a conceptual point of view, it has been shown to be NP-hard [24]. This is because the process of determining the optimal Hamiltonian Circuit requires examining every possible combination of round-trip routes either explicitly or implicitly. When the number of nodes is relatively small, an exhaustive search is a feasible means of determining the most efficient route that visits each node exactly once. Yet each time that the number of nodes in the graph, n, increases by one, the search space of the problem domain increases by a factor of $(n+1)$; this is because each possible Hamiltonian Circuit in the previous graph must now incorporate the new node, which (assuming a fully connected graph) can be inserted in $(n+1)$ possible locations. Thus, the complexity of the search space is bounded from above by $O(n!)$, making the TSP intractable once the number of nodes reaches a sufficient size [21].

The TSP is easily mapped to a number of real-world problems. One of the most obvious mappings of TSP is in terms of transportation, whether it is in minimizing the distance traveled by aircraft between various airports or in plotting the optimum routes for parcel pickups. There are, however, a number of counterintuitive uses of

the TSP in real-world applications. In the semiconductor manufacturing industry, TSP solvers provide a means of developing efficient pathways between components on an integrated circuit. Moreover, TSP solvers have also been successfully applied in genome research as a means of constructing high quality radiation hybrid maps in the shortest possible time frame. TSP solutions have even been used in operations research in order to plot the most efficient pattern of punching holes in a printed circuit board or other objects, thus resulting in reduced manufacturing costs [4].

Considering the diversity of these applications, it is clear that there is a need for developing an effective and efficient means of solving TSP's of various sizes. Unfortunately, even the best branch-and-bound strategies are unable to efficiently determine an optimal solution for TSP instances containing more than several hundred nodes [34]. Consequently, a major focus of research is in the development of metaheuristic approaches that provide good or even optimal solutions in polynomial time. Ant colony optimization, which is described later in this chapter, is one of these metaheuristics, and has been shown to be highly effective for this particular problem domain.

2.2 Gridworld Problem

Like the TSP, gridworld is an NP-hard combinatorial optimization problem that is also easily mapped to an ant colony framework. Commonly referred to within the context of machine learning, this problem domain can be directly applied to a variety of tasks where the objective is to learn the optimal path to a predefined goal [28].

The gridworld problem is defined within the context of a discrete landscape like the one presented in Figure 2.2. This landscape is divided into three parts: open locations (white) that are traversable, obstacles (black) that are not traversable, and goals (green, with "G" label). Traveling between locations in the world is limited to discrete actions in the north, south, east, and west directions, with the obvious restriction that one cannot move off of the world or travel through an obstacle. Furthermore, the cost of moving from one location to another can vary depending on

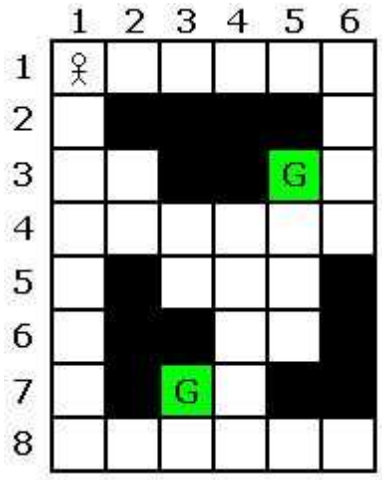

Figure 2.2: A Sample Gridworld Problem.

the direction (i.e. it costs more to travel to a particular location from the north than from the south, etc). For this thesis, however, costs are kept uniform.

Given an arbitrary starting location in an open location, the objective of the gridworld problem is to find the most efficient path to a goal location. In order to provide a more formal explanation, let a gridworld be described as a coordinate plane with R rows and C columns. Specific locations in the gridworld, then, can be denoted as $(x \in C, y \in R)$. A solution to the gridworld problem is considered to be a path P which consists of moves m_{ij} from location i to j such that the path originates from the starting location, ends at a goal, and only goes through open locations. Assuming that a cost of c_{ij} is associated with each m_{ij}, the objective function for the gridworld problem can be defined as the following:

$$MIN\ P : \sum_{m_{ij} \in P} c_{ij} \tag{2.2}$$

Based on the example provided in figure 2.2, the paths {(1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (6, 2), (6, 3), (5, 3)}, {(1, 1), (1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (4, 4), (5, 4), (5, 3)}, and {(1, 1), (1, 2), (1, 3), (1, 4), (2, 4), (3, 4), (4, 4), (5, 4), (5, 3)} all represent the shortest path from starting location (1,1) to the closest goal, as they are all of length nine.

A common extension of the gridworld problem is to determine the optimal location from every open location to a goal. This modified objective can be achieved through the repeated use of equation 2.2 on every open location, and is formally expressed as:

$$\forall_{x \in C} \forall_{y \in R} \left((x, y) \text{ is open } \Rightarrow \; MIN \; P : \sum_{m_{ij} \in P} c_{ij} \right) \tag{2.3}$$

The gridworld problem is considered to be difficult to solve due to the sheer number of possible paths that exist from any starting location. As mentioned in the previous section, determining an optimal solution requires that all other possible solutions be examined either implicitly or explicitly. Unfortunately, like the TSP, the number of possible solutions/paths in the gridworld increases exponentially as the size of the problem increases linearly. For the purposes of complexity analysis, consider a gridworld with no obstacles and exactly one goal location. Given a sufficiently large landscape, there are four possible paths from which to choose from for a given location (one for each direction). Based on this observation, the total number of paths can then be approximated as 4^l, where l is the length of the path and the 4 is the number of actions that can be made from each location. Thus, the search space of the gridworld problem is bounded from above by order $O(4^{R \times C})$ for each starting location, assuming that no path needs to visit the same location twice and can therefore be no longer than the total number of locations in the problem (since there must be at least one goal in the environment). By extension, the search space of the entire world is then bounded by order $O(R \times C \times 4^{R \times C})$. Although this complexity is reduced somewhat by assuming uniform distances between locations and the presence of walls, the fact remains that the search space for the gridworld problem is non-polynomial.

Although Equation 2.2 is similar to the objective function of TSP presented in Equation 2.1, there are several key differences between the two problem domains that clearly distinguish one from the other. Unlike in the TSP, where solution construction

is a simple matter of choosing n edges that form a Hamiltonian Circuit, generating a solution for the gridworld problem is a relatively uncertain process that involves moving to various locations until a goal is eventually encountered. Furthermore, because the TSP can be represented as a fully connected graph, it is possible to ensure that there is never a situation in which a non-starting node must be visited twice within the same solution. Unfortunately, while visiting the same location twice is undesirable in gridworld, prohibiting repeated visits is not feasible due to the possibility of becoming stuck in a dead end (due to an inability to see more than one grid cell ahead) and having to backtrack. The most significant difference between the TSP and gridworld, however, is that the former describes a single goal environment while the latter can contain multiple goals. Although there may be several Hamiltonian Circuits in a TSP instance whose total cost satisfies Equation 2.1, the problem as a whole is aimed towards the single goal of finding an optimum circuit cost. This contrasts sharply with a gridworld domain where paths terminate at just one of possibly many different but equally valid goal locations.

Thus, while the TSP and gridworld problems may share some similarities, the differences between the two are significant to the point where focusing on one or the other prevents a thorough examination of the types of optimization problems which are known to exist. For this reason, the research conducted in this thesis is considered within the context of both problem domains.

2.3 Ant Colony Optimization

As previously mentioned, NP-hard problems such as TSP and gridworld are difficult to solve to optimality due to the sheer number of candidate solutions that must be considered either explicitly or implicitly. Unfortunately, the inability of brute force approaches to solve anything larger than trivial problems has fueled the development of faster algorithms that can produce good results without the need for an exhaustive search. This quest for improved efficiency has caused many researchers to turn their focus to biological systems such as insect swarms for inspiration, as

nature has demonstrated an innate ability to devise solutions to problems using simple algorithms/rules.

Ant Colony Optimization (ACO) is perhaps one of the most successful integrations of a natural system to an algorithmic framework. First proposed by Dorigo and Gambardella [16], ACO solves problems by mimicking how real ants instinctually discover the optimum path from their colony to a food source. Rather than examining the entirety of a search space, ACO algorithms are characterized by their inclination to explore only those areas where they believe an optimum solution is likely to appear. Doing so, however, necessitates a tradeoff between optimality and speed. Because ACO is a metaheuristic approach, one cannot guarantee that the solutions produced by the algorithm are optimal. Yet according to studies conducted by Dorigo and others, ACO algorithms have been shown to generate solutions that are comparable, if not equivalent, to those generated by their brute-force counterparts [6]. Considering that these algorithms also run in polynomial time and can therefore be used on larger problem sets than is possible using a deterministic strategy, it is clear that the potential of ACO is simply too great to ignore.

The ACO framework is based off of the idea that multiple, relatively simple "ant" agents can solve complex problems through the cooperative sharing of information. The key to this collaborative process is a *pheromone* trail (τ) which simulates the chemical trails produced by real ants to mark paths of interest. As shown in Figure 2.3, the behavior of ACO during the first several time steps is largely driven by heuristic cues due to the lack of pheromone trails. As solutions are uncovered, however, the most promising ones are identified and their corresponding paths are reinforced. These trails then play a role in influencing future time steps, as ants consider paths with higher pheromones to be more desirable (Figure 2.4). The accumulation and decay of pheromones trails over time steps directly corresponds to a colony's knowledge of the problem, as they indicate the parts of the search space the ant colony believes are worth examining and which should be ignored. Given enough time, the creation of new pheromone trails will eventually come to a halt as the ants

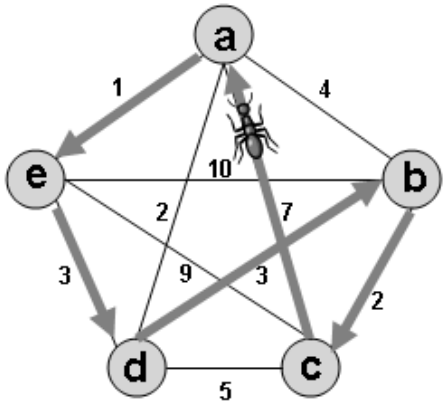

Figure 2.3: Example of ACO in the TSP During the First Time Step. Due to the lack of pheromone trails, the ant constructs a solution by picking the shortest edge from each node.

converge on a final solution and continue to reinforce the same path repeatedly (Figure 2.5). The amount of time can vary depending upon the specific implementation (a general algorithm is provided for the reader's reference in Algorithm 1), but it is not uncommon for ACO algorithms to run for hundreds or even thousands of time steps before achieving convergence [6].

Algorithm 1 Ant Colony Optimization (Generic) [18]

```
1: Initialize Pheromone Values
2: for Number of Time Steps do
3:    for Each Ant in Colony do
4:       Place Ant in Random Starting Location
5:       while Solution Not Yet Created do
6:          Add to Partial Solution
7:       for All Pheromone Values Corresponding to Solution do
8:          /* Local Update: Decrease the Value by a Percentage (ρ) */
9:          τ_solution = τ_solution × ρ
10:   for All Pheromone Values do
11:      /* Evaporation: Decrease the Value by a Certain Percentage (ρ) */
12:      τ = τ × ρ
13:   for All Pheromone Values Corresponding to Good Solutions do
14:      /* Global Update: Increase the Pheromone Value (Δτ) */
15:      τ_solution = τ_solution + Δτ
```

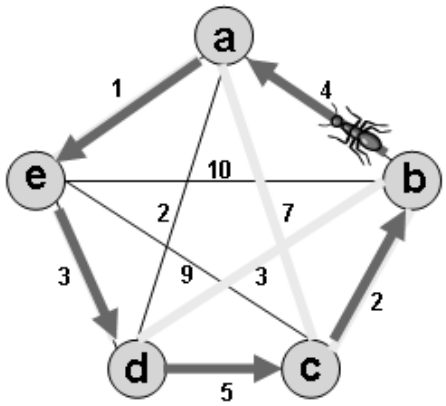

Figure 2.4: Example of ACO in the TSP Prior to Convergence. Note that the pheromone trails from Figure 2.3 are present and guide future ants' decisions. To encourage exploration, though, ants sometimes make random choices. In this case, doing so has led to the discovery of a higher quality solution.

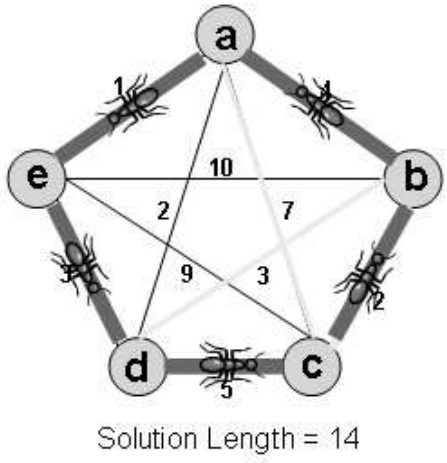

Figure 2.5: Example of ACO in the TSP at Convergence. At this point, the ants have settled on a final solution, and reinforce it constantly. All other pheromone trails fade away since they are no longer being used.

As stated in [6], there are four components that are inherent to all ant colony algorithms. In order to provide a more formal description of the ACO framework, each will now be discussed in detail.

1. *Heuristic Desirability.* Although ACO's long term behavior is strongly influenced by the presence of pheromone trails, the algorithm also relies on heuristic cues in order to make well-informed decisions. Heuristic information is especially useful during the first several iterations of an ACO algorithm, as the colony is unable to rely upon pheromone trails in order to guide the ants towards promising paths. Yet even in later iterations, research has shown that heuristics can continue to be used to great effect [5]. In order to strike a balance between the importance of pheromone trails and heuristic cues, ACO makes use of the parameters α and β. These parameters represent the relative weighting of importance between pheromone and heuristic values, respectively, and are used to influence how much influence either has in the decision making process. Since they tend to evaluate problem specific information, no general form of a heuristic can be provided. Some common examples for the TSP, however, include the nearest neighbor, minimum spanning tree, and inverse distance heuristics. Each of these are described in [5] and have been used to great effect.

2. *Transition Rule.* In order to promote a healthy exploration of the search space, ACO algorithms rely on a stochastic transition rule that guides ants from one state (i.e. a node in the TSP, or a grid cell location in gridworld) o the next. This rule, as generalized in Equation 2.4, utilizes a random value q and threshold value q_0 (both between 0 and 1, inclusively) to determine whether an ant should make a decision greedily or probabilistically.

$$NextState = \begin{cases} GreedyChoice, & \text{if } q \leq q_0 \\ ProbabilisticChoice, & Otherwise \end{cases} \tag{2.4}$$

By relying on probabilities instead of a purely deterministic approach, the transition rule gives ants the opportunity to exploit immediately promising (greedy) paths while still allowing them to occasionally explore new parts of the search space. This in turn helps the algorithm avoid the pitfall of becoming stuck in local optima.

3. *Constraint Satisfaction.* A typical characteristic of ACO calls for each ant to generate a feasible solution at the end of every time step. To enforce this characteristic, a constraint satisfaction policy is utilized which ensures that ACO is capable of providing a viable solution regardless of the number of time steps that have elapsed. Obviously, developing effective constraints for an ACO algorithm requires a thorough knowledge of the problem domain for which the algorithm is being mapped to. In the case of the TSP, the constraints are designed such that no ant travels along the same edge twice or visits a non-starting node more than once per time step. Likewise, the constraints for gridworld include checking to make sure that ants never travel past the edge of the landscape or occupy a grid cell that contains an obstacle. Depending upon the problem being solved, developing an effective set of constraints can be a difficult process. Adding to this challenge, though, is ensuring that the constraint function does not simultaneously restrict the behavior of the ants to the point where they are unable to effectively explore the search space (i.e. not allowing ants to enter an enclosed area in a gridworld problem for fear of backtracking).

4. *Pheromone Update Rule.* Developing an effective and efficient policy in regards to managing pheromone trails is perhaps the single most important aspect of any ACO algorithm. For this reason, it is important to choose a pheromone update rule that effectively captures an ant colony's knowledge of the search space. Although a review of the current literature indicates that there are a number of different pheromone update rules, most can be categorized into one of three types. The *global pheromone update rule* (line 15 of Algorithm 1) is the most common and is used at the end of time steps in order to reinforce paths

that lead to the most promising solutions. This is typically done by adding an amount of pheromone proportional to the quality of the solution (In the TSP, the reciprocal of the circuit length is used as the amount of pheromone). In the *local pheromone update rule* (line 9 of Algorithm 1), pheromones along a path are reduced whenever an ant travels on it. By doing so, it is hoped that other ants working later in the time step are less likely to choose this path, thus resulting in a wider exploration of the search space. Finally, through the *evaporation update rule* (line 12 of Algorithm 1), pheromones along all paths are reduced at the end of a time step, with the rationale being that doing so will cause the ants to ignore less promising branches of the solution space. A cursory examination of ACO algorithms such as the Ant Colony System [15], MAX-MIN Ant System [29], and Win or Learn Fast Ant System [7] indicates that while the global update rule is technically the only rule that is required, incorporating all three can lead to increased overall performance.

Since the behavior of ACO is highly influenced by the presence or absence of pheromone trails, its behavior can be naturally divided into two phases [22].

- In the *exploration* phase, ants are willing to examine various portions of the search space regardless as to whether or not that portion is known to contain high quality solutions. This phase generally occurs during the first few time steps, as the relative lack of pheromone trails allows the ants to consider a wide range of paths from which to choose from.
- In the *exploitation* stage, ants become extremely sensitive to pheromones and only make small deviations from the paths laid out in previous time steps. Not surprisingly, this phase occurs well after exploration, when pheromone trails are abundantly present and highly concentrated.

While a cursory glance at these two phases might suggest that one dominates the other, the truth of the matter is that ACO equally depends on both exploration and exploitation in order to produce high quality solutions. Without a sufficient amount

of exploration, an ant colony is unable to determine the portions of the search space likely contain good solutions. Furthermore, without sufficient use of the exploitation phase, the colony does not closely examine the solutions uncovered by exploration in order to find even better ones. Thus, in order maximize the effectiveness of the ACO technique, it is clear that a balance must be maintained between the amount of time spent searching for new solutions and the amount of time spent refining them.

Although this thesis is primarily concerned with the application of ACO algorithms in regards to the TSP and gridworld problem domains, it should be noted that flavors of ACO have been created to solve a wide variety of theoretical and real-world applications. A review of the current literature indicates that the ACO framework has been applied to a number of classical problems such as the quadratic assignment problem [31], multidimensional knapsack problem [12], and graph coloring problem [11]. In addition, a number of real-world applications, such as job-scheduling [35], vehicle routing [14], and adaptive network routing [3] have also directly mapped to the ACO. In each case, the use of an ACO algorithm was found to result in solutions that were as good or better than a robust, state-of-the-art algorithm, thus demonstrating the validity of this technique on both a theoretical and practical level.

2.4 ACO Parallelization Strategies

The ACO algorithm lends itself for use in a parallel environment. Under this parallel framework, a single large colony is broken into a set of smaller colonies (consisting of one or more ants), each of which are asked to generate solutions independently of the other. At predefined intervals, the results (i.e. solutions, pheromone values, etc) of these individual runs are then gathered and shared with one another through the use of a specialized *synchronization mechanism*. The use of multiple small colonies as opposed to a single large one makes it possible to take advantage of the stochastic nature of the ACO technique. If left to their own devices, each colony explores a different portion of the search space. Synchronizing then allows each colony to learn from the experiences gathered by others without having to explicitly gain this infor-

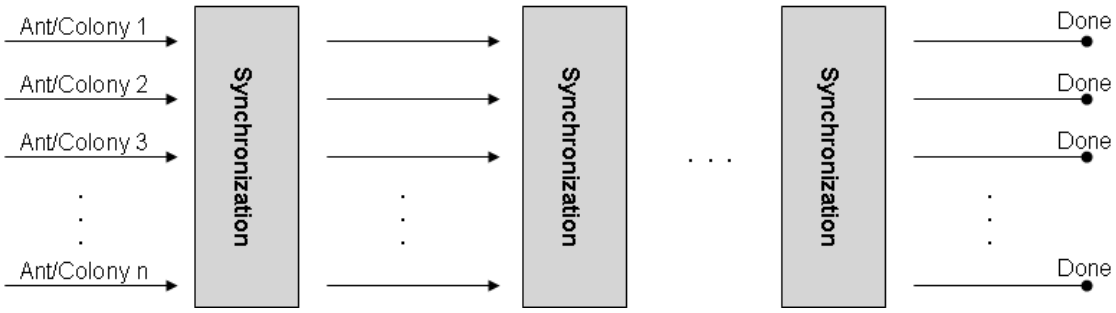

Figure 2.6: Visualization of the General Parallel ACO Strategy

mation for itself. Thus, through the use of parallelism, each processing unit is able to learn a larger portion of the search space than it does on its own, which in turn should improve its chances of generating high quality solutions in a shorter number of time steps. Figure 2.6 provides an overview of the parallel ACO strategy.

While the overarching concept of parallel ACO is promising, there are a number of unanswered questions regarding how to maximize its effectiveness. What follows in this section is a discussion of the key issues involved in the parallelization of ACO, and the various approaches that have been utilized in order to address them.

2.4.1 Number of Processing Units. One of the key aspects of parallelizing ACO concerns the number of processing units that are needed to yield good performance. While intuition might suggest that "more is better" experimental results indicate that this is not always the case. According to [10], the optimal number of processing units for a parallel ACO algorithm depends largely upon the size of the problem. Given a large TSP problem, utilizing significant numbers of processing units (i.e. 5 to 10) have been shown to be beneficial due to the fact that each colony tends to focus on a different part of the search space. As a result, each colony has a unique set of experiences which it can share with the rest of the network via synchronization. When given a small problem, however, each colony tends to explore a similar portion of the search space. In this case, having multiple processing units actually hurts performance, as colonies waste time sharing redundant information to one another. The results of Cioni's study are profound, as it suggests that parallelization is not a

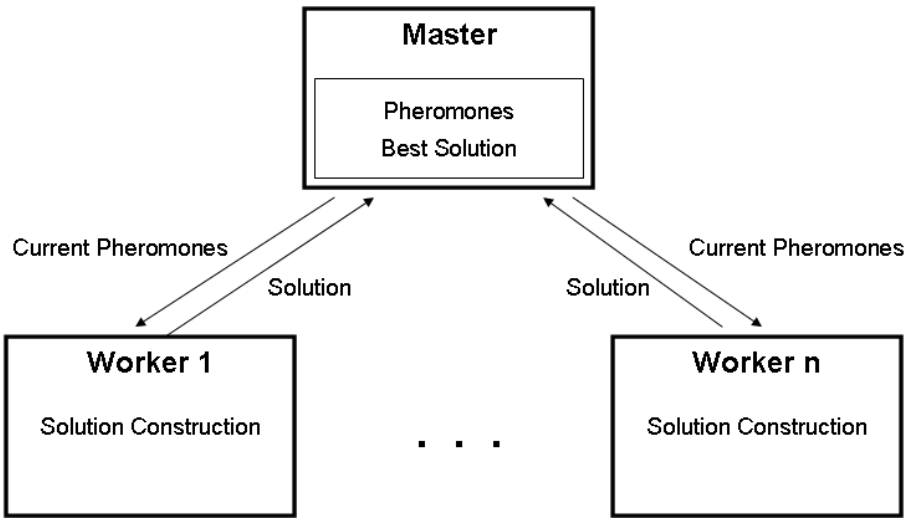

Figure 2.7: ACO Master/Worker Hierarchical Network Topology.

panacea [10]. While it is true that ACO stands to benefit from parallelization, this study indicates that these benefits only arise when the environment is large enough to take advantage of multiple colonies. Without such an environment, having many small colonies is no better than having a single large one.

2.4.2 Network Topology. At the heart of every parallel ACO implementation is a network structure that facilitates communication between processing units. An examination of the current literature indicates that a variety of network topologies have been successfully incorporated into the existing ACO framework [8]. Unfortunately, while there are obvious tradeoffs to each layout, the impact of these topologies on parallel ACO has yet to yield a clear-cut winner.

One of the most straightforward network topologies, involves the use of a hierarchical organization of processing units [32]. This organization, illustrated in Figure 2.7, divides the available processing units into a set of *workers* (representing either a colony or an ant) led by a single *master*. During runtime, the master is responsible for keeping track of the best solution found and for updating the pheromone matrix at the end of each time step. Workers, on the other hand, are only responsible for creating a solution when directed to by the master. Note that this topology utilizes a shared

memory architecture whereby the master stores information about the problem being solved and workers ask for that information as needed. Because each worker relies on the master to maintain pheromone trails, the advantage of having multiple workers is focused more on decreasing the amount of time that it takes to construct solutions during a time step than it is in sharing information between colonies.

The master/worker topology has been augmented in [13] and [30]. In the former, computational efficiency is improved through the incorporation of efficient memory usage and algorithm optimizations. In the latter, a multi-layer hierarchy consisting of several lower level masters reporting to a single overarching master is proposed and implemented. Regardless of these enhancements, though, the fundamental limitations of the hierarchical topology remain. Because workers depend on the master (or masters, in the case of [30]) in order to update the pheromone trails, synchronization must occur at the end of every time step. According to [10], this constant need for information sharing between colonies causes communication time to dominate runtime performance. Another shortcoming inherent to the hierarchical topology is the fact that workers must be given the most recent pheromone values at the start of each time step. This in turn leads to significant network overhead due to transmission delays and the quantity of data being transferred, further decreasing efficiency. Most importantly, because there is only a single pheromone matrix, the master/worker topology fails to take advantage of the unique experiences that are encountered by each worker over the course of a trial. As a result, rather than gaining more knowledge in a shorter period of time, as is one of the key motivations for parallelizing ACO, the results produced by this topology are strikingly similar to those generated by a nonparallel ACO algorithm. In fact, the only notable difference is that the parallel implementation is able to take advantage of the additional computational resources of its multiple workers to produce these results faster. When one considers the time penalties associated with the synchronization process, however, it is unclear as to how significant this advantage actually is.

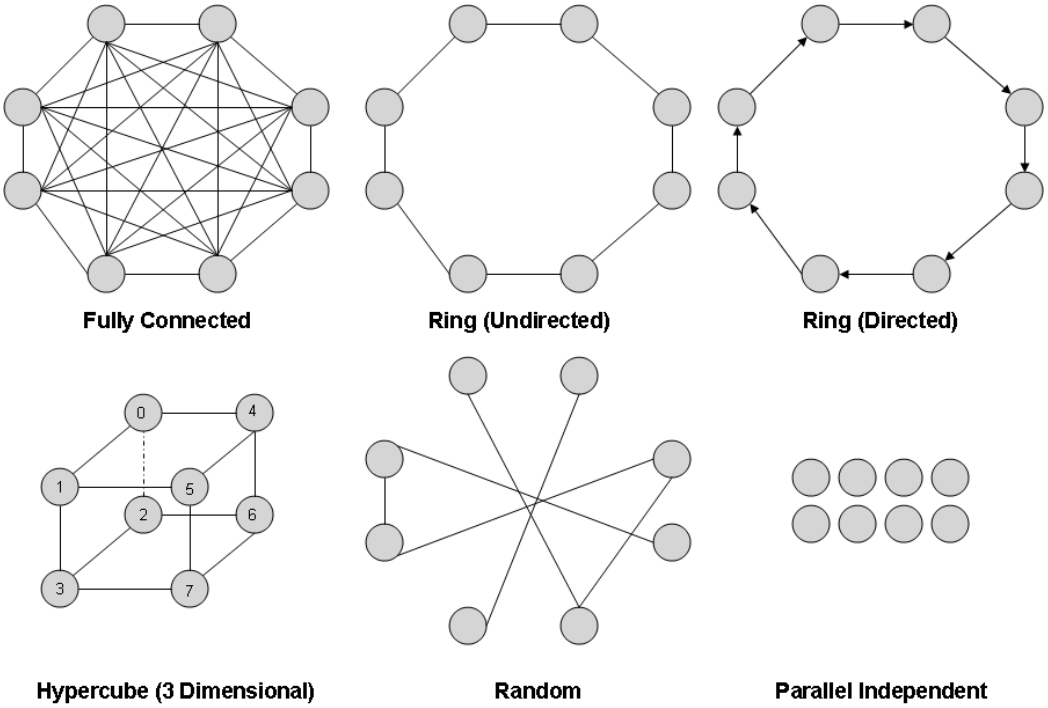

Figure 2.8: Depiction of Various Network Topologies (Assuming 8 Colonies).

Given the inherent performance bottleneck associated with the centralized master/worker scheme, some researchers have turned their attention to the creation of decentralized network topologies. Rather than maintaining a single pheromone matrix, decentralized topologies allow for the creation of multi-colony systems where each colony maintains its own set of pheromone trails. This in turn allows each colony to operate independently of one another until synchronization occurs, significantly reducing the overhead of frequent communications between colonies. A survey of decentralized topologies provided by [18] and [23] lists a variety of decentralized networks that have used in parallel ACO. Each of these networks are described below and are visualized in Figure 2.8:

- *Fully Connected.* In a fully connected network topology, each ant colony has a direct line of communication with every other colony in the network. Consequently, this topology represents the most flexible network design possible. Typically, the use of a fully connected topology requires that a single colony be

responsible for evaluating the performance of all other colonies in order to determine which ones have performed the best/worst; the results of this evaluation is then broadcasted to all other colonies in order to facilitate the synchronization process. This colony is designated the *master* of the network, although this moniker is quite different from the master/worker framework described earlier since the role is only used during synchronization and can technically be fulfilled by any colony.

- *Ring.* In a ring topology, each colony is connected to exactly two neighbors. The resulting topology restricts communications by ensuring that information can only be sent in a circular direction. The two most common forms of ring networks utilize either a directed or undirected communication scheme. In the directed ring network [23], colonies can only transmit information to one neighbor and receive it from the other. This contrasts with a non-directed ring network [18], where a colony can both send and receive information from both of its neighbors.

- *Hypercube.* The hypercube topology, as described by [18], can only be implemented when the number of colonies (n) can be expressed as $n = 2^k$, where k is an positive integer. If each colony is identified with a unique binary string, then a connection between two colonies will exist if the IDs of the two differ by only a single bit. The resulting network, therefore, is k-dimensional.

- *Random.* In a random network, connections between colonies are arbitrarily placed and are modified at the end of each synchronization step. As a result, this network is the most unpredictable network topology described thus far.

- *Parallel Independent.* Although technically not a network, the idea of simply running an isolated instance of ACO on each processor is proposed in [23] as an alternate means of organizing a set of ant colonies. The justification behind a parallel independent "network" is based on the fact that ACO is a stochastic search process. As a result, some runs of the algorithm will undoubtedly

produce better results than others. By prohibiting communication between individual colonies, the parallel independent strategy proposes that high quality solutions can be generated without the need for synchronization. Needless to say, this topography eliminates much of the overhead associated with sending and receiving data across multiple processing units, since the only data that needs to be transmitted occurs at the beginning (i.e. the problem to be solved) and end (i.e. the best solution found) of a run.

According to studies conducted by [23] and [25], the choice of network topology appears to have a significant impact on the performance of parallel ACO. In terms of execution time, communication heavy topologies such as the fully connected network suffer from the large amounts of identical data that is transferred and processed by each colony at each synchronization step. In order to minimize this time penalty, the Middendorf, Reishle, and Schmeck suggest the use of a directed ring network, which benefits from the fact that each colony can only communicate with its immediate neighbors [25]. Yet while these studies prove that communication latencies can be reduced through the smart selection of network topology, none of the layouts presented in this discussion match the efficiency of a parallel independent network. As noted in [23], parallel independent networks have a distinct runtime advantage since they completely omit the synchronization process. Fortunately, while runtime performance is important, the motivating factor behind parallelizing ACO is focused less on execution time and more on solution quality. It is impossible to entirely eliminate the time penalty associated with synchronization without removing the communication step altogether. Consequently, the best that can be expected from a true parallel ACO implementation is a topology that facilitates improved results in a comparable amount of time.

Yet while the promise of increasing solution quality through the use of an effective network layout makes the study of various topologies well worth the time and effort, the results of [23] and [25]'s studies indicate that the benefits of such research have yet to be realized. In addition to its faster runtime performance, the use of

parallel independent runs results in higher quality solutions than is achievable using any of the aforementioned network topographies. Furthermore, when examining the exploration of a parallel ACO algorithm, the results of [23] demonstrate that ants often converge prematurely when they are connected to one another over a network; parallel independent colonies, on the other hand, were shown in the same study to explore a larger portion of the search space in the same amount of time. Certainly, these findings discredit the idea of parallelizing ACO altogether. However, it is still early to make such a determination for certain.

Although there does appear to be a relationship between network topography and performance, the nature of this relationship is obscured by a number of other factors which likely play a more key role. These factors include

1. Determining what information should be shared between colonies
2. Determining how to effectively merge that information (i.e. the synchronization strategy used)
3. Determining the frequency of communication between colonies

Each of these factors undoubtedly play a crucial role in parallel ACO with regards of runtime performance and solution quality. Unfortunately, as is discussed in the following sections, researchers (including this one) are just beginning to look at these issues.

2.5 Synchronization Strategies

Up to this point, the concept of synchronization has been largely abstracted for the sake of simplicity. The importance of the synchronization step to parallel ACO, however, necessitates a closer examination of this process. As illustrated in [18], developing an effective synchronization strategy requires careful consideration as to how information can efficiently and effectively be shared between multiple colonies. What follows in this section is a review of recent work on these issues.

2.5.1 Determining what Information to Share. Although the idea of synchronization sounds good in theory, determining exactly what information is worth sharing remains a largely unresolved issue. Given that ant colonies store knowledge in the form of pheromone trails, one obvious approach is to simply allow each colony to share its entire pheromone matrix. In [20], this strategy was tested on a variety of network topologies. Unfortunately, the authors discovered that the sheer size of the pheromone matrix results in an unacceptable communication overhead, especially when the frequency of synchronization is high and the problem being solved is large. Based on this observation, researchers have turned their attention away from the direct sharing of the pheromone matrix, electing instead to focus on the exchange of smaller and more meaningful packets of data between colonies.

2.5.1.1 Migrants. One possibly useful piece of knowledge worth sharing between colonies is the solutions (whether they be tours as in TSP or paths to a goal as in gridworld) themselves. Using this approach, each colony keeps track of its solutions and determines which ones, known as *migrants*, are worthy of being shared during synchronization; typically, the best solution (i.e. shortest distance path in TSP) is chosen. These migrants are then used to reinforce other colonies' pheromone tables as determined by ACO's global update rule. The key strength of this approach is that a migrant can be represented in a fraction of the space needed to represent an entire pheromone table. As a result, sharing migrants minimizes amount of data being transferred while simultaneously ensuring that the data remains meaningful.

Manfrin, Birattari, Stutzle, and Dorigo propose a global solution exchange whereby each colony receives a copy of the best migrant discovered thus far over the entire network [23]. Another study, conducted by Middendorf et. al., calls for the sharing of the best solution discovered by each colony [25]. Finally, in [18], a strategy calling for each colony to only share and receive solutions from its nearest neighbors is proposed.

Although the results of these three studies are not directly compared to one another, an examination of their respective findings still provides some meaningful information. As observed in [23], sharing the global best solution causes parallel ACO to stagnate due to the fact that each colony updates its pheromones based on the same solution. This in turn causes each colonies' pheromone trails to closely resemble one another, which causes them to explore and exploit the same portion of the search space. Likewise, the sharing of the local best solutions between colonies has similar effects, although they are diminished somewhat due to the larger set of data that is provided to each colony. The most significant findings, according to Middendorf, result from the use of the third strategy. Since migrants are only shared between immediate neighbors via this approach, each colony receives a different set of migrants during the synchronization step. This allows the colonies to maintain fairly diverse pheromone matrices, which consequently leads to increased exploration and improved solution quality over other information sharing approaches.

2.5.1.2 Pheromone Vectors. Although the exchange of migrants is by far the most popular approach to sharing information, another equally valid strategy calls for an exchange of important pheromone values between colonies. To accomplish this, a strategy proposed in [9] uses a *pheromone vector* to transmit key pheromone values (i.e. the highest values in the matrix) between colonies. While this strategy is shown to be an effective means of running parallel ACO for the set covering problem, the approach was also found to have the same limitations as the migrant strategy. Given too large a pheromone vector, each colony's tables converged to a similar configuration, resulting in decreased exploration of the search space. On the other hand, too small a vector and the benefits of synchronization are outweighed by the increased computational complexity involved in communicating these values.

The findings obtained thus far indicate that determining what information to exchange between colonies necessitates a balance between two extremes. Since the point of exchanging information is to improve the knowledge base of each colony,

the information to be shared must be meaningful. At the same time, though, the selected information must also prevent each colony from focusing on the same part of the search space. As of this writing, the best information sharing strategies are those that give each colony a unique but equally valid set of data for use during synchronization [25]. It should be noted, however, that even the best information sharing strategies to date have yet to result in solutions that are of higher quality than those obtained through parallel independent runs. Thus, more research needs to be done in this area.

2.5.2 Determining How to Incorporate Shared Information. In addition to deciding what information should be shared, another important question in parallel ACO is determining what each colony should do with this information once it is received. Yet while this section describes a number of possible approaches, no dominant policy has emerged.

A survey of the current literature indicates that there are three common strategies used to merge shared information in parallel ACO. The first, and most popular approach is an *elitist* strategy [25]. The elitist approach requires each colony to evaluate the quality of information shared by others through the use of a fitness function (i.e. the length of a tour in the TSP). Only the highest scoring (or elite) pieces of information are incorporated into that colony's pheromone matrix. Although this strategy intuitively makes sense, the results of [23] indicates that this approach has a significant shortcoming. Since each colony has access to the same set of shared information, the elitist strategy causes each colony to reinforce their pheromone tables in the exact same way. This in turn causes the entire network to converge on the same portion of the search space, which ultimately defeats the purpose of running ACO in parallel. In order to address this problem, the authors of [25] suggest giving each colony a slightly different set of information at each synchronization step. Another approach, proposed by [10], calls for the use of different pheromone update rules for each colony. This ensures that no two colonies update their pheromone matrices in

the exact same way, even if these updates are based on the same set of information. Unfortunately, while both approaches are shown improve the performance of the elitist strategy, the quality of solutions generated through this approach are still found to be inferior to those generated through parallel independent runs.

In contrast to an elitist approach, where only the best pieces of information are used, an alternative synchronization strategy calls for the use of every piece of shared information [18]. This approach maximizes the amount of knowledge that can possibly be gained through synchronization. Unfortunately, because this method does not utilize a fitness function, each colony is updating its pheromone trails based on nothing more than blind faith that its sister colonies are all sharing meaningful information. Given the stochastic nature of ACO, though, this is highly unlikely in the average case [18]. As shown by Middendorf, utilizing every piece of shared knowledge at synchronization causes the best performing colonies to contaminate their pheromone matrices with the information provided by the worst. The end result, then, is both decreased runtime performance (due to the fact that more information is being utilized at each synchronization step) and solution quality throughout the entire network.

Given the limitations of incorporating either too much or too little information into a colony's knowledge base, the authors of [33] propose a hybrid approach that attempts to achieve the best of both worlds. Similar to weighted strategy sharing [28], this approach calls for colonies (or "clans") to assign a weight value w $(0 \leq w \leq 1)$ to the relative importance of information (in the form of pheromone values) provided by other colonies. This weighting, then, is used to update each colony's pheromone matrix as:

$$\tau_{new} = \tau_{shared} \times (1 - w) + \tau_{current} \times w. \tag{2.5}$$

Through this weighted averaging of pheromone values, the clan approach allows each colony to utilize the information shared by the entire network while simultane-

ously diluting the contaminating effects of bad colonies. Unfortunately, because w is a tunable parameter value, maximizing the effectiveness of this approach requires some trial and error. In an effort to eliminate this step, a similar weighted strategy algorithm is proposed where the value of w varies for each colony depending upon the quality of solutions found by the remainder of the network [17]. In both studies, though, the impact of this technique remains the same. Through an analysis of this technique in the TSP, both authors discovered that while solution quality can improve by as much as 0.5% through the use of this approach, it also reduced quality by the same amount once the size of the problem became sufficiently large (i.e. more than 100 nodes). As a result, it is unclear as to whether or not the benefits of this weighted approach offsets the computational resources required to implement it within parallel ACO.

2.5.3 Determining When to Synchronize. Given the emphasis placed on determining what to synchronize and how to synchronize it, it is easy to overlook the importance of determining when to invoke the synchronization process. According to recent studies, this aspect plays a crucial role in parallel ACO. As noted in [8], the rate at which colonies exchange information can have a significant impact on runtime efficiency. Furthermore, in terms of solution quality, the results of [23], [25], and [30] indicate that the performance of ACO is also influenced by when information is shared. Considering that these two areas of performance are of key importance in this thesis, it is clear that this issue deserves further consideration.

A survey of recent research efforts indicates that there are two timing strategies, or *synchronization schedules*, utilized in parallel ACO.

- The first of these schedules calls for the use of synchronization at the end of every time step. Assuming a flawless information sharing strategy, synchronizing once per time step provides a high rate of information sharing between colonies, which consequently should lead to higher quality solutions in a shorter amount of time. However, the use of this schedule has not yet produced these

results. Because synchronization requires communicating information to one or more colonies, there is a significant runtime penalty that is incurred due to network transmission delays. In fact, given a large enough network, it is shown in that a colony spends more time transmitting information than it does solving the problem when using this schedule [23]. In addition, when looking at the findings from the studies listed above, not one has found evidence suggesting that synchronizing every time step actually leads to improved solutions. On the contrary, solution quality was actually found to decrease by up to 8% through this approach [23].

- Rather than swap information at the end of every time step, an alternative approach calls for synchronization after a fixed number of time steps have elapsed. With a large enough interval (i.e. 5 time steps), this *asynchronous schedule* can be shown to significantly improve runtime performance [8]. Furthermore, in terms of solution quality, the use of synchronization sparingly also benefits parallel ACO, as it allows colonies to diversify their exploration/exploitation of the search space [25]. Yet even though the performance of parallel ACO is greatly improved through the use of an asynchronous schedule, the results of [23] indicate that using this strategy still produces lower quality than those generated with parallel independent runs. Thus, while this schedule appears to be a step in the right direction, there is still much room for improvement.

Although the research to date has concentrated on scheduling synchronization at the end of each time step or at predefined intervals, it should be noted that there are a number of alternative approaches which have been suggested throughout the years. In [18], the authors propose a schedule that differs for each colony, thus resulting in a wider variety of information that is exchanged at synchronization. Another approach proposes a more dynamic synchronization schedule in which colonies only share information once they have made a significant discovery [25]. To this date, both schedules have yet to be examined as exhaustively as the ones described above. Yet

considering that the current methods have yet to produce the desired performance, it is difficult to imagine this being the case for much longer.

2.5.4 Final Thoughts. Through this discussion, it is clear that the parallelization of ACO is an ongoing area of research. While some strides have been made in determining an effective number of colonies to use as well as how these colonies interact with one another across a network, other aspects of this process, including what to synchronize (Section 2.5.1), how to synchronize (Section 2.5.2), and when to synchronize (Section 2.5.3), have largely eluded research efforts to date. Given this current state of affairs, the rest of this thesis is focused on the study of the Area of Expertise learning strategy. The AOE technique represents a potentially revolutionary means of parallelizing ACO for three reasons. First, it proposes a novel and arguably more precise means of determining *what* information is worth sharing. Secondly, it utilizes a far more sensible means of determining *how* this information should be shared between colonies. Most importantly, by varying the time step in which this method is invoked, it is possible to exploit the findings of the previous section and determine *when* the most opportune time to synchronize is. In each respect, the AOE approach provides a fresh new perspective to the idea of parallelizing ACO. As a result, it is an approach that warrants further study.

2.6 Area of Expertise Learning

The Area of Expertise (AOE) learning strategy [1] is a novel means of sharing knowledge between multiple agents working in tandem. Cooperative learning strategies to date have centered around the idea of giving agents information and forcing them to incorporate it into their knowledge base. In a dramatic paradigm shift, the AOE technique calls on agents to develop a sense of how well they have learned the problem space. This knowledge, then, constitutes their "expertness." Through synchronization, each agent evaluates the expertise of others and adds their knowledge

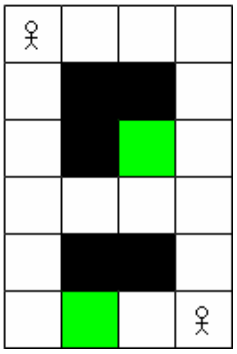

Figure 2.9: Example Gridworld Problem with Agents.

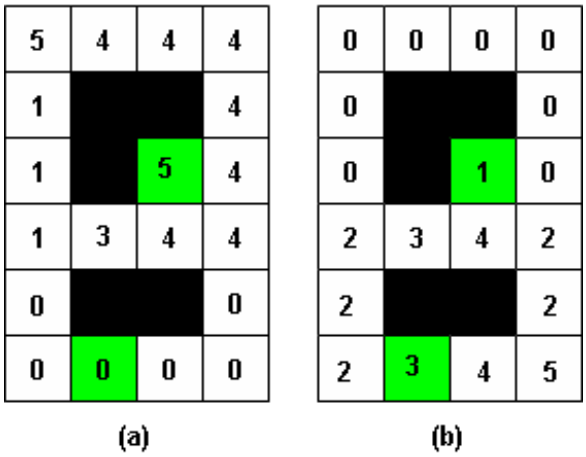

Figure 2.10: Example Visit Table for the Upper Left Agent (a) and Lower Right Agent (b).

to its own wherever it feels that it is not an expert. By doing so, each agent is able to augment its knowledge base rather than overwrite it.

In order to provide a detailed explanation of the AOE strategy, this section provides a pedagogical example involving a simple gridworld environment (Figure 2.9). For the purposes of simplicity, only two agents are considered, although the AOE approach can technically accommodate any number of agents so long as at there are at least two. Furthermore, each agent starts at opposing ends of the gridworld in order to ensure a healthy exploration of the entire landscape.

With this setup in place, each major phase of the AOE process can be summarized as follows:

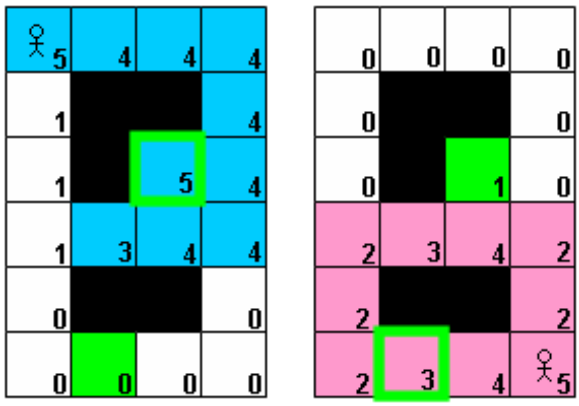

Figure 2.11: Areas of Expertise of Two Agents through Self-Determination

1. Each agent examines its own knowledge base in order to develop a sense of its expertness in the search space.
2. Agents train a classifier in order to identify the expertness of others.
3. Agents share and incorporate other agents' knowledge into their knowledge base.

Each of these steps are now discussed in detail.

2.6.1 Determination of Self-Expertness. The first step in the AOE process calls for each agent to determine where it possesses expert knowledge. To accomplish this task, each agent possesses a data structure known as a *visit table* [1]. The visit table keeps track of the number of times an agent finds itself in a particular state; in the gridworld example provided in Figure 2.9, this idea of state directly corresponds to a grid cell location. Over the course of a trial, agents traverse the search space and update their visit tables accordingly. The end result of this table after several time steps is shown in Figure 2.10.

The idea of expertness in the AOE strategy is centered on the idea that agents become experts in states that they frequently visit. By taking the median of a visit table, the authors of [1] define an *expertness threshold* which an agent can use in order to determine its AOE (shown in Figure 2.11). In Figure 2.10, the visit table values are purposefully designed such that the median of the table (3 for the leftmost agent and 2 for the rightmost) is nonzero for both agents. Had the median of the visit table

been 0, though, an acceptable alternative would be to set the threshold to 1 in order to prevent the agent from considering itself expert throughout the entire search space.

Since the visit table approach does not take solution quality into account, an obvious criticism of AOE is that it is arbitrary in its determination of expertness. It is important to remember, however, that the term *expertness* does not always have a positive connotation. According to [1], an agent is an expert when it can accurately predict the effects of its actions from a given state. At times, the agent learns what actions lead to positive consequences (i.e. knowing in gridworld where the nearest goal is from a particular cell). Undoubtedly, though, there are times when the agent only knows what actions will lead to negative consequences (i.e. knowing in the TSP that traveling along a certain edge is too costly to be part of an optimal solution). In both cases, the agent has experience in these locations that may prove useful to others. Thus, in both cases, the agent is considered to have expert knowledge.

2.6.2 Parzen Classification. Once each agent has determined its AOE, it must then evaluate the expertness of others. Unfortunately, this task is complicated since each agent develops its own perspective as to what constitutes expertness. As a result, determining the expertise of other agents requires looking at their knowledge base from this unique perspective. In [1], the authors propose the use of a Parzen classifier which is specifically trained to recognize expert knowledge from a given agent's perspective. Parzen classification is a nonparametric technique which approximates a function through the use of density calculations [27]. Given a set of data points residing in n-dimensional space and a query point, q, the Parzen technique works by constructing an n-dimensional hypercube about q and sampling the number of data points which reside within it; classifications, then, are based upon the number of data points which reside within this hypercube.

Within the context of [1], the data points for Parzen classification correspond to the knowledge base of the agent that is using it. The contents of this knowledge base are already designated expert or nonexpert based upon the contents of its cor-

responding visit table, and are used by the classifier to learn what an agent considers expert knowledge. Query points, on the other hand, consist of those pieces of knowledge (obtained from other agents) that need to be evaluated for expertness. Since knowledge in [1] is one-dimensional (i.e. Q-table values), and classifications requires a binary "expert" or "nonexpert" response, the Parzen classification technique for evaluating expertness is relatively simple, and is described in detail by Algorithm 2.

Algorithm 2 Parzen Classification of Expertness [1]

1: **for** A given query point q **do**
2: Retrieve the agent's knowledge base and determine its expertness (as per Section 2.6.1).
3: Define a hypercube of length h and volume h^n in n-D space centered about the point that represents q
4: Determine
k_c = the number of expert samples (N_e total) inside the hypercube
k_n = the number of nonexpert samples (N_n total) inside the hypercube
5: Calculate
$P_e = \frac{1}{h^n} \times \frac{k_e}{N_e}$
$P_n = \frac{1}{h^n} \times \frac{k_n}{N_n}$
6: **if** $P_e \geq P_n$ **then**
7: Return Expert
8: **else**
9: Return Nonexpert

An analysis of Parzen classification indicates that there are two potential shortcomings to this approach. Since a typical gridworld problem may only consist of hundreds of locations, the Parzen classifier described above must learn to distinguish between expert and nonexpert states from an unusually small training set. This in turn raises serious doubts as to whether or not the classifier can make accurate predictions of expertness. Furthermore, as seen in Algorithm 2, the key to the Parzen classifier rests in determining the optimal dimensions of the hypercube. If the cube is too large, then the classifier will believe that the agent has expert information in nearly every location of the gridworld. When the cube is too small, however, the classifier will not believe that agents possess any expert knowledge. Given the undesirability of either extreme, the authors of [1] rely on empirical analysis in order to

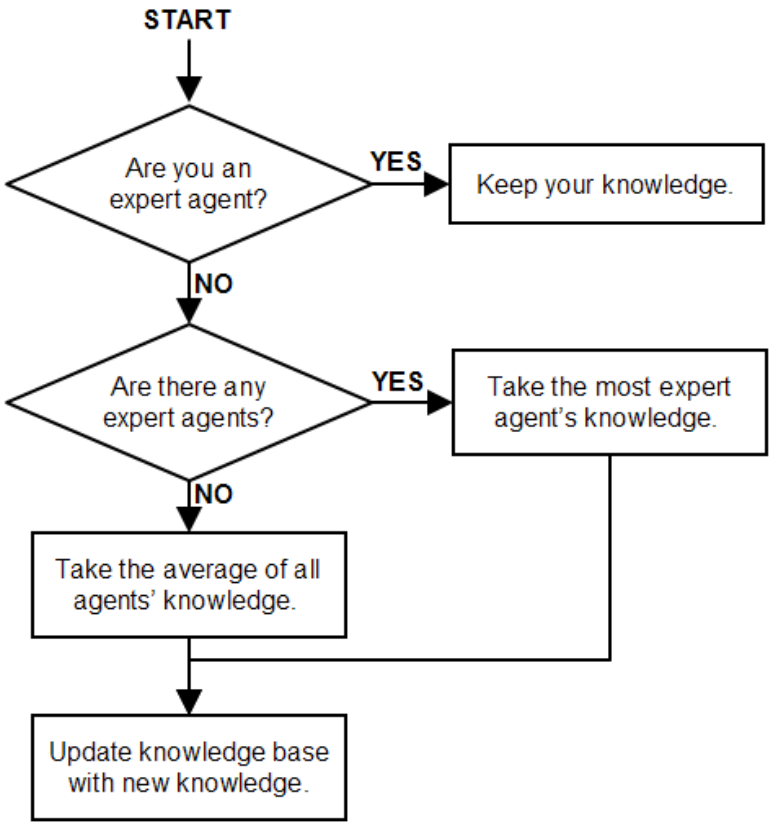

Figure 2.12: AOE Synchronization Decision Chart. Note that this Chart is Followed by Every Agent for Every Piece of Nonexpert Data in its Knowledge Base.

determine the best hypercube dimensions. Unfortunately, these dimensions only work for the authors' Q-table values, and need to be redetermined for other representations of the knowledge base (which in the case of this thesis takes the form of pheromones).

Despite the concerns listed above, it can be shown that the use of a Parzen classifier can result in predictions of expertness that are similar and comparable to those produced by the visit tables. These results, in conjunction with the elegance and simplicity of the classifier, informally verify the effectiveness of this strategy. As a result, this thesis also utilizes the Parzen method.

2.6.3 Performing Synchronization. The final step of the AOE strategy calls for the synchronization of knowledge across multiple agents. To do this, each agent relies on the decision making process presented in Figure 2.12. For each location

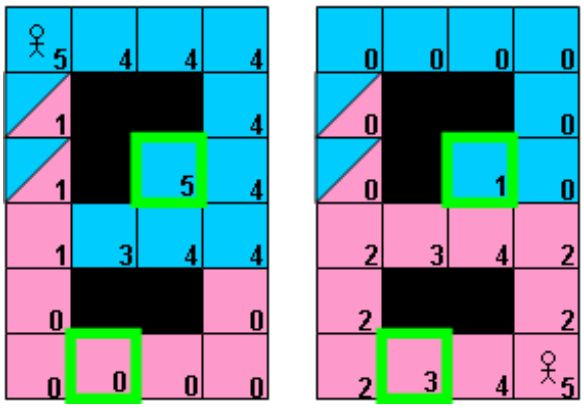

Figure 2.13: AOE of Two Agents After Synchronization. Note that Locations with Two Colors Denote that an Average of Both Agents' Knowledge was Used.

in the gridworld, the agent determines whether or not it already possesses expert information. If it does, then the agent makes no change to its knowledge base. When the agent lacks expert information, however, it evaluates the expertness of other agents at that particular location (using the Parzen classifier). If other agents' AOE include this location, then the agent incorporates the experience of the most expert agent into its knowledge base. Otherwise, a simple average of each agent's knowledge is used to replace the existing value. The synchronization process is complete once each agent has had a chance to compare its knowledge with others and make changes as necessary. Assuming a perfect classifier, an example of the knowledge bases of multiple agents after synchronization is provided in Figure 2.13.

The novelty of the AOE approach stems from the fact that each agent retains its expert knowledge throughout the course of its lifetime. The purpose of the synchronization step, then, is not to overwrite an agent's knowledge base, but to simply augment nonexpert areas with the experience gathered by other agents. By doing so, it is possible to give each agent a more complete understanding of the search space, which should help it make more informed decisions as to which portions of the solution space are worth searching. More importantly, by ensuring that each agent retains its own expert knowledge, it is possible to ensure that no two agents have the same knowledge base. This attribute is perhaps the most important advantage of the

AOE learning strategy, as different areas of expertise undoubtedly lead each agent to explore the search space in a slightly different manner. This in turn leads to a wider exploration which hopefully results in the discovery of higher quality solutions.

Although the AOE strategy has only been implemented in terms of robots, it does not take a significant stretch of the imagination to envision a similar learning scheme with regards to parallel ACO. The reason for this is because the agents in [1] behave similarly to ant colonies, and the use of Q-table values can easily be substitute with pheromone values [19]. Considering the difficulties encountered thus far in getting parallel ACO to yield quality results, it is clear that the potential of this strategy is simply too great to pass up. As a result, the main focus of this thesis is in regards to incorporating AOE learning into the ACO framework.

III. Methodology

This chapter describes the methodology used to incorporate Area of Expertise Learning within the Ant Colony Optimization framework through a bottom-up approach. Following a general overview, the first step in the design process calls for the examination of a specific flavor of ACO known as the Ant Colony System (ACS), which has already been successfully applied to the traveling salesman problem in [15]. Using this algorithm as a template, an ACS algorithm designed specifically for gridworld is then created. Afterwards, a description of the modifications needed to parallelize ACS is provided. Finally, the chapter concludes by showing how the AOE learning technique can be integrated into parallel ACO.

3.1 Overview

The methodology described in this chapter facilitates the parallelization of ACO by means of a processor farming model (commonly referred to as a master/worker hierarchy) implemented on a local area network. Through this approach, a single large ant colony is partitioned into a predefined number of *worker sub-colonies* (consisting of one or more ants) which each reside on a different processing unit. A dedicated *master* process, then, oversees the entire network and serves as a central point of communication. Figure 3.1 provides a visual representation of this topology.

As is typical in a distributed architecture, colonies are purposefully isolated so that each can operate independently from the rest of the network. In order to support the use of AOE learning, however, this isolation is interrupted at times to allow for swapping of expert knowledge. As seen in the UML sequence diagram in Figure 3.2, inter-colony communications are implemented via a handshaking process between workers and the master. To initiate this process, each worker transmits a request message to the master, along with the information it wishes to share (i.e. pheromone values). Once the master receives a request from the entire network, it then combines this information via the AOE process and returns a unique set of

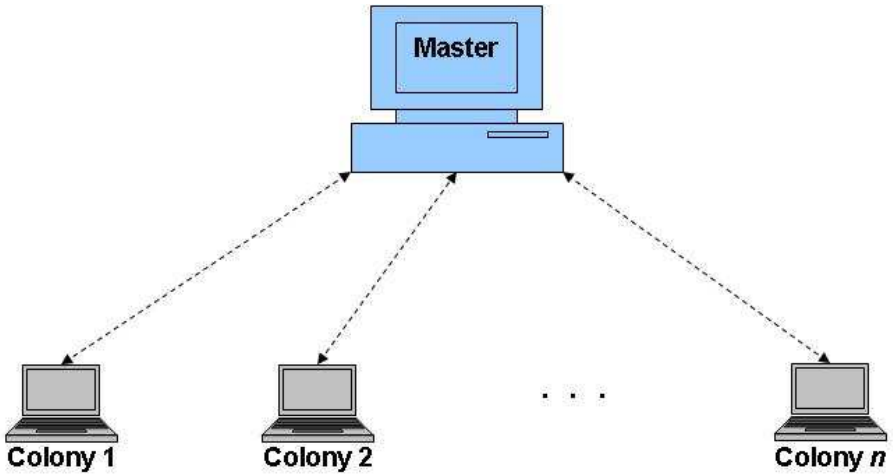

Figure 3.1: A Description of the Master/Worker Topology Used Throughout this Chapter. Note that the Arrows Signify Two-Way Communications.

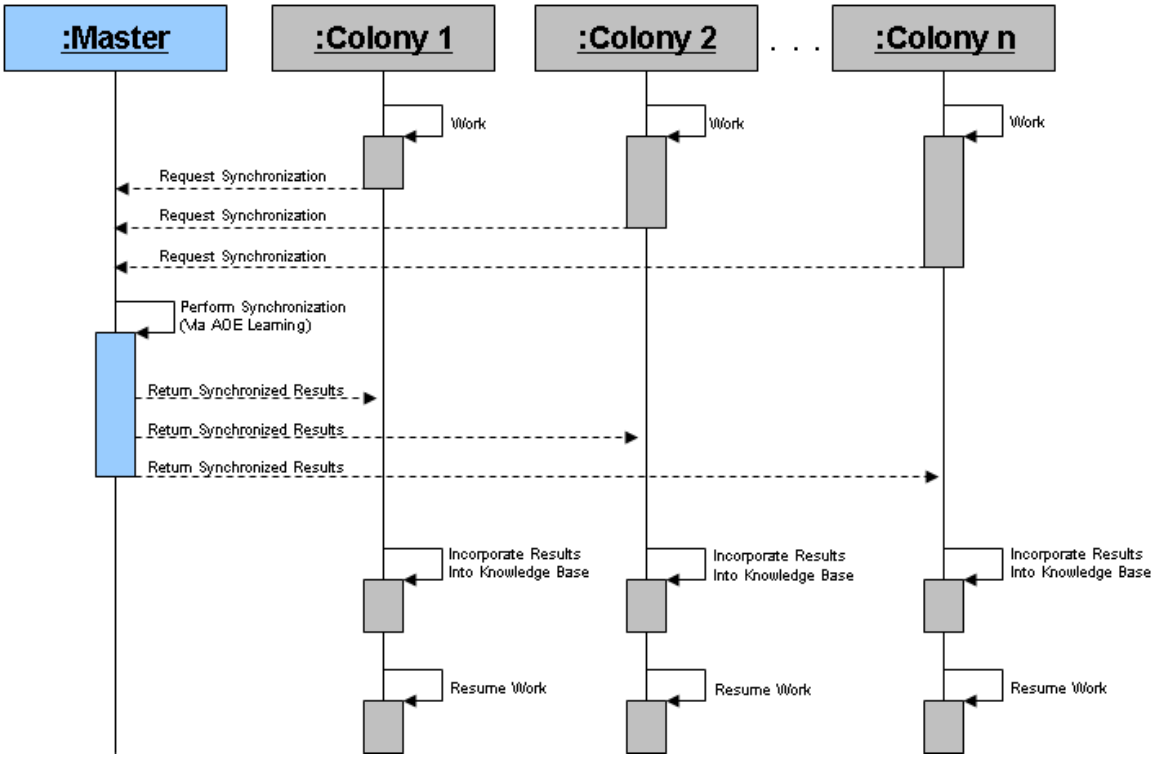

Figure 3.2: High-Level UML Sequence Diagram of the AOE Learning Process.

results to each participant. After updating their knowledge bases with these results, the colonies resume working from where they left off.

Although the methodology described above is intuitively simple, there are numerous aspects of this design which have been intentionally abstracted for ease of explanation. What follows in this chapter is a detailed discussion of each of these aspects.

3.2 Design of Algorithms

Figure 3.2 introduces the idea that colonies are responsible for performing some set amount of work. For the purposes of this thesis, this notion of work takes the form of an ACO algorithm. This section provides a detailed discussion of the two ant colony algorithms used in this thesis. The first, ACS-TSP, is well-established and commonly used throughout the academic community. ACS-GRIDWORLD, on the other hand, represents a brand-new ant colony algorithm created specifically for this research. Although these two algorithms are technically different, both share a number of fundamental similarities in terms of data structures used and overall behavior. This ensures that any modifications made to one algorithm (as proposed in the following sections) can be laterally applied to the other as well.

3.2.1 ACS-TSP. The ACS-TSP algorithm, developed by Dorigo et al. [15], is often regarded as the textbook application of ACO to an NP-hard problem domain. As illustrated in Algorithm 3, the structure of ACS-TSP can be broken down into four main phases. In the *initialization phase*, the time step counter (t) is set to 0, pheromone values (represented as a matrix, τ) along every edge are set to a nonzero value (τ_0), and ant agents are assigned a starting city/node either randomly or uniformly. In addition, the algorithm creates a sorted data structure known as a *candidate list* (cl) which keeps track of the closest cities available from each node; this is done as a time-saving measure to prevent the algorithm from having to sort the set of all edges from each node by cost (c) when determining where to travel

to next. While the size of the candidate list is an adjustable parameter, ACS-TSP strictly enforces that the contents of this list only contain those nodes which the ant has yet to visit. By doing so, the algorithm ensures that the list only contains those moves which are legal for the ant to make.

Although the importance of the initialization phase cannot be overstated, the crux of the ACS-TSP algorithm rests in the exploration and exploitation of the search space. To accomplish this task, the algorithm relies on a *solution construction phase* whereby each ant in the colony is given the opportunity to completely solve the problem before the next is allowed to proceed. Prior to the start of its turn, each ant obtains a local copy of the candidate list which it can modify independently from the rest of the colony. With this list in hand, the ant constructs a solution in the following manner:

If the city where the ant is located (i) has a nonempty candidate list ($cl \neq \phi$), the ant generates a random number q between 0 and 1. This random value is then compared to q_0, which is a predefined threshold value. When q is less than or equal to q_0, the ant selects a destination city ($j \in cl$) such that the product of τ_{ij}^{α} and η_{ij}^{β} (where τ_{ij} is the pheromone present on the edge between nodes i and j, $\eta_{ij} = \frac{1}{c_{ij}}$, and α and β are weights that indicate the relative importance of selecting destinations based off of pheromone values or cost, respectively) is maximized (upper part of line 16). If, on the other hand, q is larger than q_0, the ant picks a destination city in a stochastic fashion, using the formula described in line 18 of Algorithm 3 to calculate the probability of traveling to each node; this equation takes into account the pheromone concentrations (τ_{ij}^{α}) and heuristic desirability of traveling to a node (η_{ij}^{β}) versus those of any other city that has yet to be visited (contained in the set J_i^k). In the event that the candidate list is empty, though, the ant simply makes a greedy choice and selects the node closest to its current location (line 21). Once an ant has made a decision, it removes the selected node from all candidate lists ($cl = cl - j$), updates J_i^k ($J_i^k = J_i^k - j$), and sets its current location to j ($i = j$). The entire

decision making process is then repeated until a valid solution (consisting of a path going through all nodes exactly once) is formed [6].

As seen in line 25 of Algorithm 3 (where ρ represents the decay factor), a *local pheromone update phase* occurs every time an ant travels from one node to the next. Instead of serving as a reinforcement mechanism, however, the purpose of this local update is to decrease the amount of pheromone present along the most recently traveled path. Since ants are attracted to paths with large pheromone values, local updates decrease the probability that other ants will travel along the same path. This in turn encourages a broader exploration of the search space within each time step.

Once each ant has constructed a solution, the algorithm compares each one to the best known tour (T^+) and its length (L^+), replacing the two whenever a higher quality solution has been found by the colony. Following this, the algorithm then enters a *pheromone reinforcement phase* where the pheromones along each edge in T^+ are increased according to the equation in line 33. The pheromones along all edges are then evaporated slightly (decreased) in order to prevent a single pheromone trail from becoming too overwhelming (line 35). Finally, t is incremented by one and compared to the maximum number of time steps, $t_{\max}$. If the two are equal, the algorithm halts. Otherwise, all ants are placed in new starting locations and a new time step begins.

According to [6], ACS-TSP has been shown to converge on high quality solutions in a reasonable number of time steps (i.e. less than 1000). Unfortunately, as is commonly the case with stochastic search techniques, ACS-TSP's performance is tied to its parameter settings. There are no fewer than five parameters (α, β, q_0, ρ, and cl) whose values directly affect the behavior of the algorithm. Yet while an obvious remedy is to simply tune these parameters to their optimum values, the results of [24] indicate that the best parameter settings for ACS-TSP vary depending upon the particular problem being solved. Thus, for the purposes of this thesis,

only the parameters described in [15] (detailed in Section 4.1), which are empirically demonstrated to result in good overall performance, are used.

3.2.2 ACS-GRIDWORLD. The adaptation of ACS to the gridworld problem domain borrows heavily from the framework provided in Algorithm 3. Due to a number of fundamental differences between the gridworld and TSP problem domains, however, several key modifications are required. These modifications are described below, and the resulting algorithm is provided in Algorithm 4.

During the initialization phase of ACS-GRIDWORLD, each element of the pheromone matrix is once again assigned a nonzero value of τ_0. Instead of representing edges as they did in ACS-TSP, though, (i, j) pairs now correspond to row and column locations (i.e. a single grid cell) within the landscape. The number of ants used is now left as a parameter value to be arbitrarily decided by the user (instead of being proportional to the size of the problem as in ACS-TSP), and the random starting locations assigned to each ant are monitored to ensure that they do not begin at obstacles or goals (line 5). Due to the fact that an ant can only move in the four cardinal directions (north, east, south, west) and can visit a location repeatedly, the use of a candidate list is no longer required.

Following the initialization phase, ACS-GRIDWORLD gives each ant the opportunity to construct a path from its starting location to a goal state using the same one-at-a-time strategy employed in Algorithm 3. To accomplish this task, the algorithm relies on a simplified solution construction phase whereby each ant either makes a greedy or probabilistic choice depending upon the values of q and q_0 (line 12); this phase is identical to the one utilized in ACS-TSP when the candidate list is nonempty. Since ants may end up retracing their steps due to backing up after becoming trapped or by random occurrence, it is impossible to predict how many moves are needed before an ant reaches a goal location. This is in sharp contrast to the TSP, where a solution is guaranteed after $(n - 1)$ moves have been made. Thus, whereas the time steps of ACS-TSP have a constant runtime complexity, the overall

speed of ACS-GRIDWORLD can widely fluctuate depending upon how quickly the ants find a goal. This in turn makes the evaluation of ACS-GRIDWORLD's runtime performance more difficult to accurately gauge.

While the disparities between ACS-GRIDWORLD and ACS-TSP have been minimal so far, the process of updating pheromone trails introduces a dramatic difference between the two. Although both algorithms utilize the same local update (line 19) and evaporation rules (line 27), ACS-GRIDWORLD reinforces the solutions discovered by every ant at the end of each time step (line 21). This is a significant departure from ACS-TSP, where only the best solution discovered thus far is reinforced. The rationale behind this new reinforcement rule is based on the widely varying objectives of the two problem domains. Unlike the TSP, where there is only a single optimal solution (or multiple solutions with the same minimum cost), the "answer" to a gridworld problem consists of a *policy* that shows the optimal path from every open location to a goal. Since there is no way of directly comparing the paths of two ants when they start at different locations, ACS-GRIDWORLD considers both of them to be equally valid, and reinforces them accordingly.

By allowing every ant to lay pheromone, ACS-GRIDWORLD is capable of learning an entire landscape in a relatively short amount of time. Unfortunately, such a strategy carries with it an increased possibility of suboptimal paths being mistakenly reinforced. To address this concern, the algorithm relies on two safety mechanisms. The first is the aforementioned random placement of ants prior to the start of each time step, which ensures a good coverage of the search space. The second feature is an enhanced pheromone update rule which removes loops and repeated cells from each ant's solution prior to reinforcement. This updated rule, shown in line 22, prevents ants from encouraging repeated visits to non-goal states, since doing so is never beneficial. Through a combination of these two mechanisms, the risk of converging on suboptimal solutions in ACS-GRIDWORLD is largely mitigated. As a result, the algorithm is expected to yield good performance in the average case.

3.3 Parallelization of ACS

In their current form, both of the ACS algorithms described in Section 3.2 are self-contained and designed to run on a single processor. As a result, several modifications are required before either can be utilized in a parallel environment like the one outlined in Figure 3.1. This section describes the parallelization of ACS-TSP and ACS-GRIDWORLD via a two-step process. The first step examines the algorithmic modifications needed to incorporate inter-colony communications within ACS. This is followed by a discussion of the network topology and communication model used to coordinate these colonies as they work in tandem.

As illustrated in Algorithm 5 (which provides a generic template which can be applied to both ACS-TSP and ACS-GRIDWORLD), the parallelization of ACS is supported through the addition of two segments of code. The first segment, shown in line 1, is a simple remote-access mechanism that allows the algorithm to be told which problem to work on by a master process. The second, and more crucial addition to ACS is a conditional statement that determines when/if a colony should share its findings with the rest of the network. This statement (line 8) is evaluated at the end of each time step, and triggers a synchronization request whenever it returns true; the algorithm then transmits the information that it wishes to share and waits for a response. Since the optimal conditions for synchronization (pheromone values, a specific time step, etc) are not yet known, the contents of this conditional statement are intentionally left vague. For the purposes of this discussion and of the subsequent chapters, this line is currently set to return true once a predefined time step (or interval of time steps) has been reached. Through the experimental process, however, it is hoped that this simple condition can be replaced with one that more accurately determines the most opportune time to synchronize.

In addition to algorithmic changes, the parallelization of ACS also requires some consideration as to the "physical" means by which individual colonies interact with one another. As noted in the overview, the communication model used in this thesis

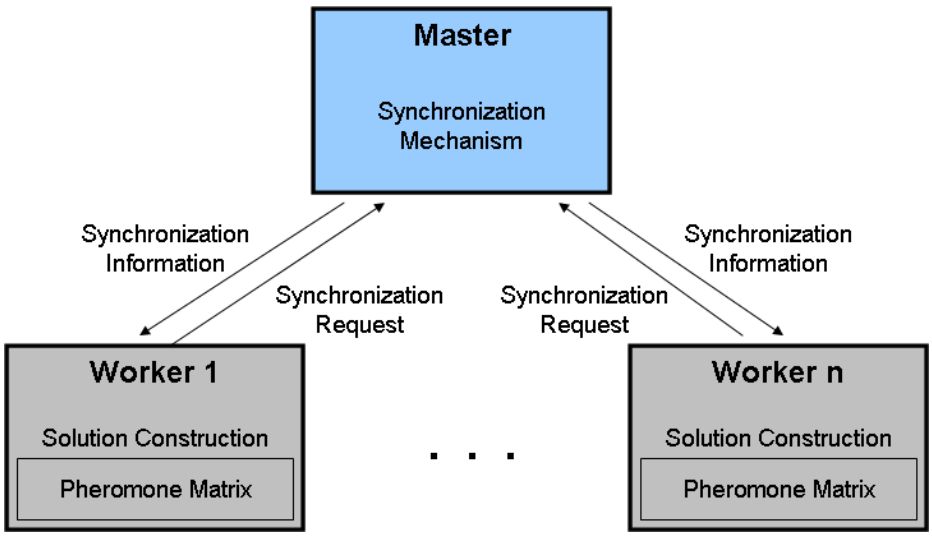

Figure 3.3: Detailed View of the Master/Worker Network Layout.

is a modified master/worker topology similar to the one described in [32]. As is typical of such a layout, each colony works independently from the rest of the network and only communicates with its master. Unlike a traditional master/worker layout, however, the hierarchy (as seen in Figure 3.3) used in this thesis is unique in that each colony maintains its own pheromone trails. Through the decentralization of the pheromone matrix, it is possible for each colony to take advantage of the stochastic nature of ACO. This allows each colony to explore and exploit different candidate solutions simultaneously. Furthermore, since workers no longer need to query the master at every time step for the latest pheromone values, this implementation also benefits from improved runtime performance due to decreased communication overhead. In its current form, worker colonies remain autonomous throughout much of the solution construction process and only communicate with the master whenever they wish to share information. The master, then, assumes the role of the synchronizer by collecting these requests, processing them, and returning the results of this process to their rightful owners. This clear delineation of responsibilities is reflected in Figure 3.3.

Through this discussion, it is clear that the parallelization of ACS is a nontrivial task. It is important to remember, though, that the modifications described in this

section are not new, and have been recreated in a variety of studies concerning parallel ACO (as described in Chapter II). Instead, the novelty of this thesis rests in the development of the synchronization mechanism, which has been purposely abstracted up to this point. As a result, the final section of this chapter focuses exclusively on the AOE learning process, and provides a means of incorporating this new form of cooperative learning within parallel ACO.

3.4 Incorporation of AOE Learning within ACS

The final step in the design process calls for the precise mapping of AOE learning into the parallel ACS framework described in Section 3.3. Whenever synchronization is deemed necessary by an ant colony (line 8 of Algorithm 5), AOE learning is used to augment that colony's pheromone matrix with the expert information provided by others. At this point, it is important to remember that the definition of *area of expertise* differs depending upon the problem domain. In gridworld, AOE naturally corresponds to particular grid cells in the landscape where an agent is considered to have compelling knowledge. For the TSP, though, AOE can be thought of as the edges where expert knowledge is contained. While these definitions may differ semantically, they are equivalent to one another syntactically. This allows for a lateral application of AOE learning to both problem domains.

The UML sequence diagram provided in Figure 3.4 provides a low-level depiction of the AOE learning process when applied to parallel ACS. What follows in this section is a breakdown of this diagram into its fundamental stages:

1. *Self Determination of Expert Knowledge.* As described in Chapter II, each colony keeps track of the frequency in which edges or grid cells (depending upon the problem domain) are visited. This visit table is updated during each execution of RunTimestep(). Once a synchronization time step has been reached (assuming that synchronization is triggered by the number of time steps executed, as noted in Section 3.3), the algorithm enters the DetermineSelfExpert-

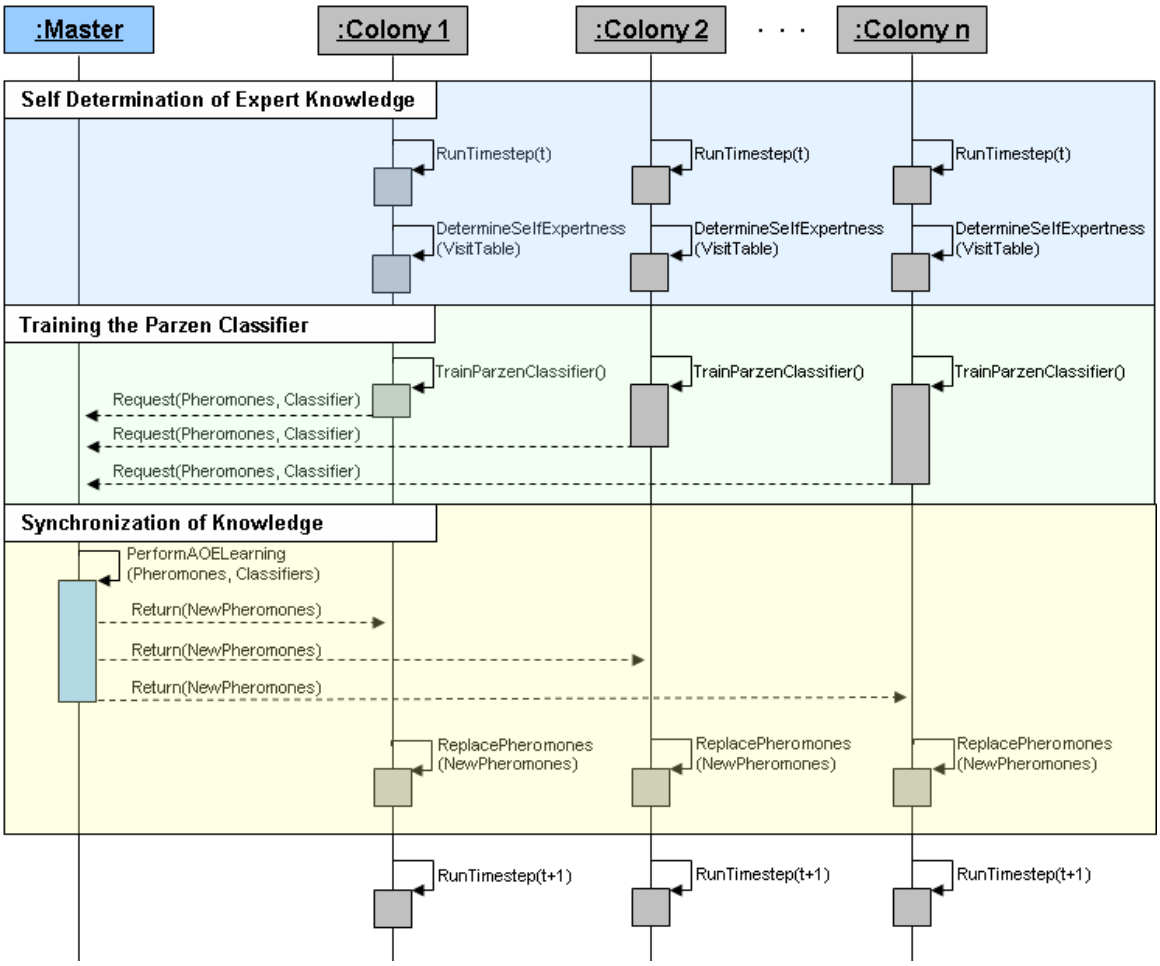

Figure 3.4: Low-Level UML Sequence Diagram of the AOE Learning Process in Parallel ACS.

ness() method, where self expertness is determined by taking the median of the visit table and using it as a threshold value. All edges or grid cells that have been visited at least this many times (as well as their corresponding pheromone values) are considered to be expert.

2. *Training the Parzen Classifier.* The AOE approach calls for each colony to train a Parzen classifier to evaluate the expertness of other colonies [1]. In order to do this, each colony uses its own visit table and pheromone matrix values as training data; only the pheromone values, however, are stored by the classifier. Determinations of expertness for any arbitrary pheromone value can then be made via Algorithm 2. As stated in Section 2.6.2, one of the key challenges in this training process rests in determining the optimum hypercube dimensions for the classifier. Since both sets of training are one-dimensional, however, this tuning process can be accomplished via a linear search. For the purposes of this thesis, the TrainParzenClassifier() method keeps track of the highest and lowest pheromone values in the training data. The difference between these two values then represents the maximum allowable dimension of the hypercube. By decreasing this value and evaluating the accuracy of the resulting classifier, it is possible to find an approximation of the best Parzen window size. A summary of this tuning process is provided in Algorithm 6.

3. *Synchronizing of Knowledge.* Once the previous two steps are complete, each worker colony transmits its current pheromone matrix and trained classifier to the master and waits for a response. The master, in turn, augments each matrix according to the AOE decision process described in Figure 2.12. At the conclusion of this process, the master sends each worker an updated pheromone matrix. With this new matrix in hand, the worker then replaces its entire knowledge base and begins work on the next time step.

The complete pseudocode for parallel ACS with AOE learning is provided in Algorithm 7 for the sake of completeness. A cursory examination of this code reveals two interesting points. First and foremost, unlike the other ACS algorithms discussed

thus far, Algorithm 7 has been divided into two parts in order to reflect the separate but complimentary roles of the master and worker as noted in Figure 3.3. The second, and perhaps most important characteristic of this pseudocode, however, is that it does not contain any problem domain specific data/control structures. This is intentionally done in order to show that methodology described in this chapter is problem independent. As a result, the general structure of Algorithm 7 can be applied to both ACS-TSP and ACS-GRIDWORLD, with each case resulting in an ant colony algorithm that is both parallelized and enhanced with AOE learning.

3.5 Closing Remarks

With the methodology complete, the focus of this thesis moves away from design and shifts towards evaluating the impact of AOE learning when combined with parallel ACO. In the following chapter, ACS-GRIDWORLD is used in order to demonstrate the proof-of-concept of AOE learning within ACO. This is followed by an examination of AOE in the ACS-TSP algorithm in order to determine how the approach works in a single objective environment. Note that this second application is of key importance, as the TSP is more representative of the types of problem domains typically associated with ACO.

Algorithm 3 ACS-TSP Algorithm [6]

/* Initialization */
for every edge (i, j) **do**
  $\tau_{ij} = \tau_0$
Generate cl for each city
for $k = 1$ to total number of cities **do**
  Place ant k on a randomly chosen city
Let T^+ be the shortest tour found and L^+ be its length
/* Main Loop */
for $t = 1$ to t_{max} **do**
  for $k = 1$ to total number of cities **do**
    /*Solution Construction*/
    Get a local copy of all candidate lists
    Build tour t^k by doing the following $n - 1$ times:
    if the cl is not empty **then**
      Choose the next available city, $j \in J_i^k$ in the candidate list as follows:

$$j = \begin{cases} argmax_{u \in J_i^k} \{[\tau_{iu}]^\alpha \times [\eta_{iu}]^\beta\}, & \text{when } q \leq q_0 \\ J, & \text{otherwise} \end{cases}$$

      where $J \in J_i^k$ is chosen according to the probability:

$$p = \frac{[\tau_{iu}]^\alpha \times [\eta_{iu}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}]^\alpha \times [\eta_{il}]^\beta}$$

      and where i is the current location
    else
      choose the closest $j \in J_i^k$
    Remove j from all local candidate lists
    /* Local Pheromone Update */
    After each transition ant k applies the local update rule:
    $\tau_{ij} = (1 - \rho) \times \tau_{ij} + \rho \times \Delta\tau_0$
  for $k = 1$ to total number of cities **do**
    if an improved tour is found by ant k **then**
      update T^+ and L^+
  /* Pheromone Reinforcement */
  for every edge (i,j) $\in T^+$ **do**
    Update pheromone trails by applying the rule:
    $\Delta\tau_{ij} = 1/L^+$
    $\tau_{ij} = (1 - \rho) \times \tau_{ij} + \rho \times \Delta\tau_{ij}$
  for every edge (i,j) $\in \tau$ **do**
    $\tau_{ij} = \tau_{ij} \times (1 - \rho)$
/* Done */
print the shortest tour and its length

Algorithm 4 ACS-GRIDWORLD

/* Initialization */
for every location (i, j) **do**
  $\tau_{ij} = \tau_0$
for $k = 1$ to total number of ants **do**
  Place ant k on a randomly chosen open location
/* Main Loop */
for $t = 1$ to t_{max} **do**
  for $k = 1$ to total number of ants **do**
    /*Solution Construction*/
    Build tour t^k by doing the following until a goal location is reached:
    Choose the next available location, $j \in J_i^k$ from the four cardinal directions:

$$j = \begin{cases} argmax_{u \in J_i^k} \{[\tau_{iu}]^\alpha \times [\eta_{iu}]^\beta\}, & \text{when } q \leq q_0 \\ J, & \text{otherwise} \end{cases}$$

    where $J \in J_i^k$ is chosen according to the probability:

$$p = \frac{[\tau_{iu}]^\alpha \times [\eta_{iu}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}]^\alpha \times [\eta_{il}]^\beta}$$

    and where i is the current location
    Remove j from all local candidate lists
    /* Local Pheromone Update */
    After each transition ant k applies the local update rule:
    $\tau_{ij} = (1 - \rho) \times \tau_{ij} + \rho \times \Delta\tau_0$
  /* Pheromone Reinforcement */
  for $k = 1$ to total number of ants **do**
    for every unique location (i,j) $\in T^k$ **do**
      Update pheromone trails by applying the rule:
      $\Delta\tau_{ij} = 1/L^+$
      $\tau_{ij} = (1 - \rho) \times \tau_{ij} + \rho \times \Delta\tau_{ij}$
  for every edge (i,j) $\in \tau$ **do**
    $\tau_{ij} = \tau_{ij} \times (1 - \rho)$
/* Done */
Print the final pheromone matrix

Algorithm 5 Parallel Ant Colony System (Generic)

```
1: Get Problem to Work On
2: Initialize Ant Starting Location and Pheromone Matrix
3: for Number of Time Steps do
4:    for Each Ant in Colony do
5:       Construct Solution
6:    Calculate Best Solution
7:    Reinforce Pheromones
8:    if Synchronization Needed then
9:       Transmit Request for Synchronization
10:      Transmit Information to Share
11:      Wait for Response
```

Algorithm 6 Determining Optimal Hypercube Dimensions

```
1: for a given pheromone matrix do
2:    max = maximum pheromone value in matrix
3:    min = minimum pheromone value in matrix
4: h_best = max − min
5: h_accuracy = accuracy of classifications for a given size (Algorithm 2)
6: n = number of hypercube sizes to try
7: tmp = h_best
8: while h_accuracy ≠ 100% do
9:    tmp = tmp − h_decrement
10:   tmp_accuracy = accuracy of classifier with hypercube size of tmp
11:   if tmp_accuracy ≥ h_accuracy then
12:      h_best = tmp
```

Algorithm 7 Parallel Ant Colony System with Area of Expertise Learning (Generic)

```
1: /* Worker */
2: Initialize Ant Starting Locations and Pheromone Matrix
3: for Number of Time Steps do
4:    for Each Ant in Colony do
5:       Construct Solution and Update Visit Tables
6:    Update Pheromones
7:    if Synchronization Needed then
8:       Determine AOE Using Pheromone Matrix and Visit Table
9:       Train Parzen Classifier (Algorithm 6)
10:      Transmit Pheromone Matrix and Classifier to Master
11:      Receive New Pheromone Matrix from Master
12: /* Master */
13: for Each Synchronization from All Workers do
14:    Wait for Each Worker to Transmit Pheromones and Classifier
15:    for Each Worker do
16:       Synchronize Pheromones (Figure 2.12)
17:       Return Updated Pheromone Matrix to Worker
```

IV. Results and Analysis

This chapter evaluates the effectiveness of AOE learning in ACO using the parallel ACS-TSP and ACS-GRIDWORLD algorithms developed in Chapter III. The first section provides an overview of the experimental setup used with regards to parameter settings, benchmarks, and test configurations. This is followed by a discussion concerning the efficiency of the AOE mechanism in terms of computational time and classifier accuracy. In the third section, the gridworld problem is focused on exclusively in order to show how the policies generated through the use of AOE learning in ACS compare to those generated in [1]. With the proof-of-concept of AOE learning validated for ACO, focus then shifts to the TSP problem domain in order to examine how the performance of ACS-TSP is affected by the sharing of expert knowledge between colonies. The chapter then concludes with a closer analysis of ACS-TSP in order to determine if an effective synchronization strategy exists.

4.1 Experimental Setup

The stochastic nature of ACS poses a number of unique challenges when it comes to developing an experimental setup that yields meaningful results. As previously mentioned, ant colony algorithms are extremely sensitive to their parameter settings and can converge on varying qualities of solutions depending upon the values of α, β, and q_0. Furthermore, without a comprehensive suite of benchmarks, it is difficult to accurately gauge the performance of the algorithm under various conditions and problem types. Finally, since no two runs of ACS are exactly the same, an effective test plan must be developed in order to develop a good understanding of the algorithm's behavior in the average case. What follows in this section is a discussion of the various design choices made with regards to ACS-TSP and ACS-GRIDWORLD in order to address each of these issues.

Rather than undergo a lengthy and inconclusive search for the optimum parameter settings, both ACS-TSP and ACS-GRIDWORLD are tuned to the values defined in [6] unless otherwise noted. Consequently, standard ant colony parameters

such as $\alpha = 1.0$, $\beta = 2.0$, and $q_0 = 0.9$, and the rate of pheromone decay (ρ) is set to 0.1. Furthermore, whenever candidate lists are used (as is the case of ACS-TSP), the size of any single list is limited to one-third of the total number of nodes in a given problem. These parameter values are not guaranteed to be optimal, but they have been frequently used in previous studies of ACO with great success [15].

Unfortunately, while the majority of ACS's parameters have been thoroughly studied in a single processor environment for the TSP, the novelty of the research conducted in this thesis necessitates a rethinking of two key areas. First, in the case of initial pheromone values (τ_0), a common strategy is to use an amount equal to the inverse of the Nearest Neighbor Heuristic [15]. While this approach works in ACS-TSP, the lack of an equivalent heuristic in gridworld necessitates that τ_0 be assigned an arbitrarily small value such as 0.1 in ACS-GRIDWORLD. Second, in order to facilitate a fair comparison of performance between single and parallel processor runs, the total number of ants used in ACS-TSP and ACS-GRIDWORLD, regardless of the number of processing units, is always equal to the number of nodes or goal locations within a problem, respectively. Note that this is different from other parallel ACO implementations in which each additional colony increases the total number of ants in the network. This restriction, combined with the strict use of the parameters described above, guarantees that any difference in performance observed through the use of parallel ACS is the result of AOE learning and not the product of some unfair advantage.

With the issue of parameters addressed, the next aspect to consider is that of benchmarks. Instead of relying on randomly generated problem instances whose quality cannot be guaranteed, both ACS-TSP and ACS-GRIDWORLD are evaluated using a series of standardized test cases provided by the academic community. For the TSP, benchmarks (along with their optimum solutions, if known) are obtained from TSPLIB [26], and are chosen such that the resulting test suite consists of graphs of various sizes and node configurations ranging from 22 to 442 nodes. In the gridworld problem domain, however, the entire test suite consists of a single gridworld problem.

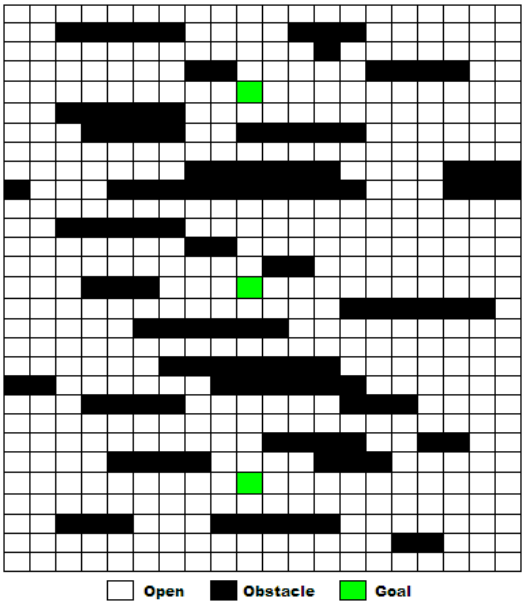

Figure 4.1: The Imanipour Gridworld Benchmark as Described in [1].

The selected gridworld (depicted in Figure 4.1) is identical to the 20×29 landscape from [1]. Thus, by focusing on the Imanipour gridworld, it is possible to directly compare ACS-GRIDWORLD's performance to the results obtained in [1].

The final impediment to creating an effective experimental setup rests in the design and execution of the experiments themselves. Since ACS-TSP and ACS-GRIDWORLD rely on randomness in order to generate high quality solutions, it is impossible to gain a complete understanding of either algorithm's behavior based on observations from a single trial. For this reason, both algorithms' average case performance is approximated through thirty independent trials of one-thousand time steps. This facilitates an analysis of ACS-TSP and ACS-GRIDWORLD through the use of first and second order statistics (assuming a Gaussian distribution). Unfortunately, the need for multiple runs, combined with the constant rerunning of tests in order to study the impact of parallelism and synchronization schedules on performance, makes this approach computationally demanding. To address this issue, both ACS algorithms were implemented in Java 1.5 in order to take advantage of the language's efficient data structures and increased portability. The resulting software was

Table 4.1: Summary of the Core Experimental Configuration Used Throughout this Chapter.

	Algorithm	
	ACS-TSP	*ACS-GRIDWORLD*
Ant Colony Parameters		
α	1	1
β	2	2
q0	0.9	0.9
τ0	1/(Nearest Neighbor Heuristic)	0.1
Number of Ants	Equal to Number of Nodes in Problem	Equal to Number of Goals in Problem
Candidate List Size	Equal to 1/3 of Nodes in Problem	N/A
Benchmarks Used		
	att48 (48 Nodes)	Imanipour (20X29)
	berlin52 (52 Nodes)	
	eil51 (51 Nodes)	
	st70 (70 Nodes)	
	ulysses22 (22 Nodes)	
	eil101 (101 Nodes)	
	pr226 (226 Nodes)	
	pcb442 (442 Nodes)	
Test Plan		
Number of Runs Per Problem	30	30
Number of Time Steps Per Run	1000	Varies
Number of Colonies Used	1, 2, and 5	1, 3

then run on a variety of platforms ranging from a network of desktops featuring dual Pentium 4 processors, 2.0 gigabytes of RAM and the Windows XP operating system, to a cluster computer featuring Pentium 3 processors running Unix. Although this approach prevents time data from being easily compared across differing hardware configurations, it is important to remember that the concept of *time* in ACO is more concerned with time steps rather than wall clock time [16]; hence, this implementation is intended to focus on the former rather than the latter. While this setup is not ideal, its ability to conduct multiple runs of ACS-TSP and ACS-GRIDWORLD in a relatively short amount of time allows for a more thorough analysis than would be possible otherwise.

A summary of the experimental setup used in this chapter as described above is provided in Table 4.1 for the sake of thoroughness. In the following sections, both ACS-TSP and ACS-GRIDWORLD are studied in an effort to determine the usefulness of AOE learning within the ACO framework. Yet while the focus of each experiment may vary from one section to the next, it is important to remember that their core

Table 4.2: Parzen Classifier Training Time (100 Window Sizes Considered, Results in Seconds).

Time Steps Before Training	Benchmark: Gridworld	Benchmark: TSP				
	Imanipour	ULYSSES22	EIL51	ATT48	BERLIN52	ST70
5	6.794	4.676	130.704	102.824	140.933	471.053
10	6.732	4.630	131.003	101.787	141.708	463.987
15	6.735	4.563	128.984	102.507	141.338	461.795
20	6.676	4.478	128.329	101.955	140.776	463.450
25	6.750	4.468	129.457	102.124	141.180	464.346
30	6.782	4.468	130.397	103.465	143.100	465.331
35	6.761	4.452	129.825	103.217	142.906	466.350
40	6.745	4.452	129.652	103.054	142.754	465.270
45	6.853	4.422	129.540	103.153	143.507	464.634
50	6.640	4.311	127.577	101.379	139.990	454.452
Average	6.747	4.492	129.547	102.546	141.819	464.067

configuration remains the same. This in turn ensures that the findings of this thesis, as described below, are both credible and accurate.

4.2 Parzen Classifier Efficiency

Before evaluating the effectiveness of AOE learning in either algorithm domain, it is necessary to consider the impact of this technique in terms of efficiency. As noted in Chapter II, the crux of the AOE strategy rests in each colony's use of a Parzen classifier to evaluate the expertness of its neighbors. Unfortunately, the lack of a known optimal Parzen window size for the gridworld and TSP domains, combined with the need for accurate classifications, necessitates that multiple hypercube dimensions be tested at each synchronization step. In this section, the AOE training process is examined using the Imanipour gridworld as well as the ulyssess22, att48, eil51, berlin52, st70 and TSP benchmarks. For each problem, the number of hypercube dimensions examined at synchronization is capped at 100. Training, then, occurs after a set number of time steps ranging from 5 to 50 (chosen arbitrarily for this section) have elapsed. Provided below are the results of these experiments in terms of both runtime performance and classifier accuracy.

4.2.1 Runtime Performance. Although classifier training is rarely a trivial task, the results of Table 4.2 serve as a startling reminder of just how time-consuming

Table 4.3: Best Parzen Window Sizes (Hypercube Dimensions) as Determined Using Algorithm 6.

		Benchmark					
		Gridworld	TSP				
		Imanipour	ULYSSES22	EIL51	ATT48	BERLIN52	ST70
Time Steps Before Training	5	2.01123	0.00474	0.00394	0.00396	0.00411	0.00401
	10	1.58475	0.00464	0.00418	0.00410	0.00447	0.00470
	15	1.63748	0.00394	0.00437	0.00412	0.00453	0.00391
	20	1.57188	0.00368	0.00418	0.00434	0.00446	0.00389
	25	1.60254	0.00401	0.00445	0.00412	0.00435	0.00387
	30	1.59113	0.00415	0.00426	0.00423	0.00425	0.00450
	35	1.46829	0.00454	0.00425	0.00431	0.00451	0.00402
	40	1.50004	0.00422	0.00411	0.00418	0.00431	0.00422
	45	1.76146	0.00423	0.00418	0.00404	0.00421	0.00422
	50	1.47696	0.00347	0.00406	0.00414	0.00412	0.00432
	Average	1.62058	0.00416	0.00420	0.00415	0.00433	0.00417

Table 4.4: Hypercube Sizes Used in this Chapter (Based on Averages from Table 4.4).

	Hypercube Size
Gridworld	1.621
TSP	0.0042

this process can be. Even by severely restricting the number of window sizes which are examined and considering only relatively small test cases, determining the dimensions of an accurate Parzen classifier is still a computationally expensive process, with training times ranging from as few as four seconds (ulysses22) to nearly eight minutes (st70) per synchronization session. While these results indicate that the complexity of training increases exponentially as the size of the problem increases linearly, a closer look at Table 4.2 also illustrates that training time is relatively unchanged with respect to each problem. This means that there is no need to take the number of time steps into consideration when selecting a synchronization schedule.

Fortunately, while the current method of Parzen classifier training is impractical for all but the smallest test cases, the results of Table 4.4 indicate that this process can be greatly sped up. According to the results of a 5% ($\alpha = 0.05$) two-tailed t-test, there is no statistically significant evidence to suggest that varying the number of time steps elapsed prior to training has any bearing on the best window size of the classifier. Moreover, while a cursory glance clearly illustrates a disparity in window sizes between TSP and gridworld benchmarks (brought on by the fact that both problem

domains utilize a different value of τ_0), closer examination of Table 4.4 reveals that the optimal hypercube dimensions discovered for each TSP benchmark are statistically indistinguishable from one another. In light of these findings, it appears that the importance of Parzen classifier training is overrated. As a result, rather than subject ACS-TSP and ACS-GRIDWORLD to this lengthy tuning process, a more efficient design choice is to simply hard-code the classifier using an empirically determined average window size for each problem domain. For gridworld, this design choice means that each classifier is initialized with a hypercube width of 1.621. TSP classifiers, then, are set to the value of 0.00420, which is the average window size across all benchmarks. Although this assumption does not guarantee an optimum window size, it does reduce the training time by at least a factor of 100 since there is no longer a need to constantly assess and compare the accuracy of each classifier after training. This in turn should give the AOE strategy the performance boost that it needs in order to remain competitive with other parallel ACO approaches.

4.2.2 Parzen Classifier Accuracy. While runtime speed is important, being able to correctly assess the expertness of others based solely on pheromone values is a key aspect of AOE learning in ACO as it directly impacts the quality of information that is exchanged between colonies during synchronization. Consequently, a thorough analysis of AOE's efficiency must also take the accuracy of the Parzen classifier into account. For the purposes of this section, each of the classifiers described in Tables 4.2 and 4.4 are evaluated using their respective training data as the validation set. A summary of these results, then, is provided in Table 4.5.

As noted in Chapter II, the Parzen classifier is designed to gauge the expertness of other colonies based on the notion of expertness of the colony that trained it. Yet despite having to work with a relatively small training set, the results of Table 4.5 indicate that Parzen classification provides a surprisingly effective means of assessing the expertness of others. With only a single exception, each of the classifiers shown in Table 4.5 are typically able to differentiate between expert and nonexpert states

Table 4.5: Average Parzen Classifier Accuracy (Expert and Nonexpert States).

Time Steps Before Training	Benchmark					
	Gridworld	TSP				
	Imanipour	ULYSSES22	EIL51	ATT48	BERLIN52	ST70
5	67.6%	84.6%	89.2%	89.1%	89.8%	89.4%
10	72.9%	82.5%	87.8%	87.0%	87.8%	87.5%
15	72.9%	81.8%	86.6%	85.3%	86.0%	86.1%
20	73.9%	82.2%	85.7%	83.7%	84.7%	84.9%
25	73.3%	82.6%	84.7%	82.7%	83.8%	83.7%
30	73.9%	82.9%	84.1%	81.6%	82.8%	83.0%
35	74.8%	82.8%	83.3%	80.3%	81.1%	81.9%
40	75.1%	83.1%	82.9%	79.7%	80.8%	81.3%
45	70.6%	83.3%	82.2%	78.8%	79.7%	80.7%
50	75.1%	83.3%	81.3%	78.0%	79.1%	80.0%
Average	73.0%	82.9%	84.8%	82.6%	83.6%	83.8%

Table 4.6: Average Parzen Classifier Accuracy (Expert States).

Time Steps Before Training	Benchmark					
	Gridworld	TSP				
	Imanipour	ULYSSES22	EIL51	ATT48	BERLIN52	ST70
5	44.7%	60.6%	72.4%	73.0%	72.8%	71.4%
10	59.3%	62.1%	73.2%	71.8%	70.0%	69.9%
15	56.7%	73.3%	71.2%	69.1%	66.8%	67.5%
20	59.3%	77.2%	70.5%	65.5%	64.8%	66.1%
25	57.7%	81.0%	67.6%	64.2%	63.5%	63.7%
30	59.2%	81.6%	67.5%	62.0%	61.9%	63.5%
35	62.1%	80.4%	65.8%	59.7%	58.1%	61.5%
40	61.9%	80.1%	65.8%	59.5%	58.4%	61.4%
45	49.4%	80.6%	64.2%	58.8%	57.0%	61.5%
50	62.7%	79.3%	63.4%	57.0%	56.4%	59.8%
Average	57.3%	75.6%	68.2%	64.0%	63.0%	64.6%

at least 70% of the time. Furthermore, these results also illustrate that the Parzen method works best within the TSP problem domain, as the accuracy of the Imanipour classifier tends to be lower than its TSP counterparts by an average of 10.5%. Most importantly, the use of a 5% t-test indicates that varying the number of time steps prior to synchronization does have a statistically significant impact on classifier accuracy from within the same problem. This suggests that changing the synchronization time step can noticeably affect the quality of information that is exchanged between colonies (and by extension, solution quality since better pheromone trails should lead to better solutions), which lends credence to the idea that an optimal synchronization schedule exists.

Unfortunately, while the above results are promising, focusing on total accuracy only provides a small glimpse of a much larger picture. In Tables 4.6 and 4.7, the

Table 4.7: Average Parzen Classifier Accuracy (Nonexpert States).

Time Steps Before Training	Benchmark					
	Gridworld	TSP				
	Imanipour	ULYSSES22	EIL51	ATT48	BERLIN52	ST70
5	91.1%	94.2%	94.8%	94.5%	94.8%	94.5%
10	86.8%	92.6%	94.0%	93.9%	94.6%	94.1%
15	89.2%	86.5%	94.0%	93.9%	94.5%	94.0%
20	88.5%	85.0%	93.9%	94.5%	94.6%	94.0%
25	89.0%	83.5%	94.4%	94.3%	94.5%	94.0%
30	88.6%	83.8%	94.1%	94.8%	94.5%	93.9%
35	87.5%	84.3%	94.4%	94.8%	94.9%	93.9%
40	88.2%	85.0%	94.1%	94.8%	94.9%	93.6%
45	91.8%	85.1%	94.4%	94.7%	94.7%	93.3%
50	87.5%	86.1%	94.2%	95.0%	94.7%	93.6%
Average	88.8%	86.6%	94.2%	94.5%	94.7%	93.9%

results of Table 4.5 are broken down in order to show the Parzen classifier's accuracy at identifying expert and nonexpert states separately. On their own, these tables reveal many of the same trends noted above. A comparison of the two, however, shows that while the Parzen classifier is capable of identifying nonexpert states nearly 90% of the time, its classifications of expertness are sometimes only slightly better than blind guessing. Also, expert accuracy generally does down as the size of the problem increases. Given such a wide disparity in accuracy, the results of Table 4.7 can be attributed to the fact that Parzen classification is extremely pessimistic, and assumes that states do not contain expert information. Yet while this bias allows for extremely accurate identifications of nonexpertness, its inability to consistently identify expert states hinders the performance of the classifier as a whole, especially as the number of time steps (and consequently, the number of expert states) increases; this trend is clearly visible in Tables 4.5. In fairness, this behavior may prove to be unintentionally beneficial since it ensures that little if any state information in being inaccurately classified as expert and shared between colonies. When considering how much expert information is mistakenly ignored, however, it is clear that there is still significant room for improvement.

Based on these results, it is clear that there are both benefits and shortcomings to the use of Parzen classification within the AOE framework. On the one hand, this approach appears to be adept at identifying nonexpert states. The key to the AOE

process, however, is in being able to identify expertness, and Parzen classification falls short in this regard. Although the accuracies noted above are impressive at times, it is important to remember that these values represent the upper bound of performance, since they are generated by evaluating the classifier using the exact same data points used to train it. Consequently, a decrease in classification accuracy is to be expected once these data points no longer resemble the training set. The bigger issue, though is whether or not this decrease is significant enough to hamper the effectiveness of the AOE strategy in ACO. The following sections shed more light on this matter.

4.3 Impact of ACO with AOE Learning in Gridworld

To demonstrate the effectiveness of information sharing in parallel ACO, the AOE technique is first considered with regards to the gridworld problem domain. This section compares how the exchange of expert knowledge in ACO compares to prior results reported in [1] for the gridworld domain using ACS-GRIDWORLD. Due to the peculiarities of gridworld, an analysis of ACS-GRIDWORLD's performance over multiple runs does not yield meaningful findings since there is no widely accepted way to calculate the mean or standard deviations of multiple policies. Consequently, this section provides a typical example of how the sharing of AOEs affects each colony's convergence to a final policy.

For the purposes of this demonstration, the AOE learning process is examined through the use of three colonies working in tandem. Each colony is forced to start in different zones of the gridworld (as shown in Figure 4.2), although movement throughout the landscape is unrestricted once an ant has been placed. Through this approach, each colony is then encouraged to develop its own unique area of expertise. This section contains a breakdown of the AOE learning process into its basic steps followed by an examination of the final policies created through synchronization.

4.3.1 Self-Determination of Expertness. As described in Section 2.6, the first step in the AOE process calls for the self-evaluation of each colony's expertness.

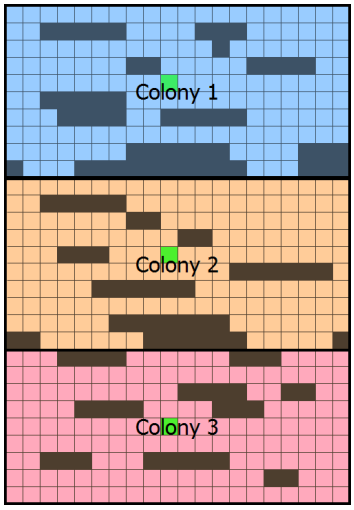

Figure 4.2: Starting Location for Each Colony.

To demonstrate this capability within ACO, each ACS-GRIDWORLD colony is allowed to run to convergence (the point in which the AOE of a colony does not change; empirically determined to occur after 25 time steps), which is defined as the point where the colony neither gains nor loses expertness with each time step. A determination of expertness for each state, then, is made using the median of each colony's visit table as a threshold value. Figure 4.3 shows a typical resulting AOE map for all three colonies when using this classification scheme.

As expected, the assignment of unique starting areas allows each colony to develop a different area of expertise than its neighbors. While there is some overlapping of AOEs, the layout of higher expertise "levels" (denoted by darker colors) seen in colonies 2 and 3 illustrates that each colony's AOE is most focused near its starting region. Note that while all three colonies possess expert knowledge for more than two-thirds of the gridworld and learn all three goal locations, Figure 4.3 also reveals that no colony is able to converge upon an effective policy over the entire landscape. This is good because it allows all three to benefit from an exchange of expert knowledge.

Although fundamental differences between ACS-GRIDWORLD and Q-learning prevent a point-by-point comparison, a cursory glance at Figure 4.4 suggests that the results obtained thus far are generally similar with those reported in [1] from a

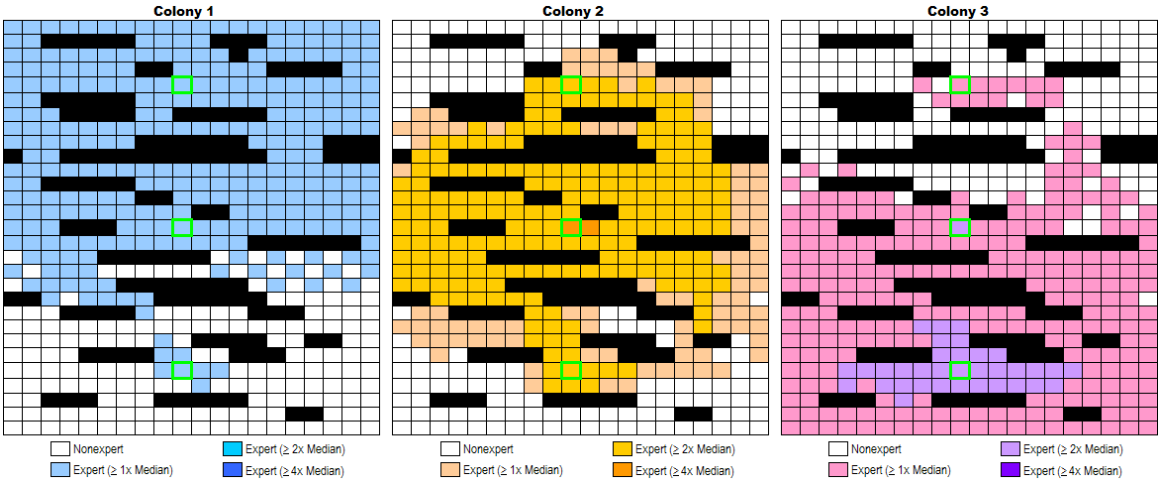

Figure 4.3: A Typical Example of Areas of Expertise as Determined through Self-Assessment (After 25 Time Steps)

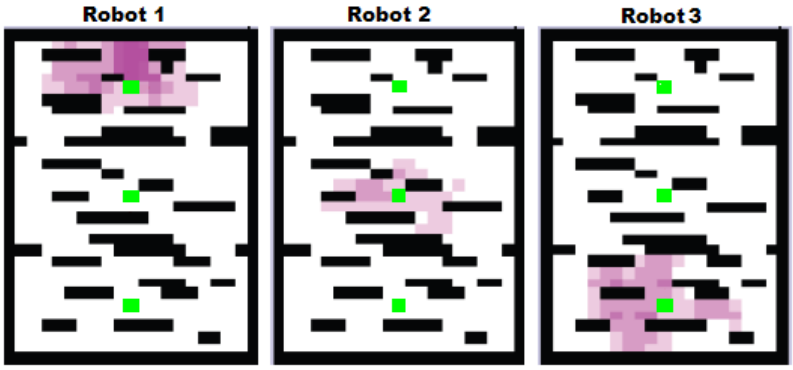

Figure 4.4: Self-Determination of Expertness as Reported in [1] using Q-Learning.

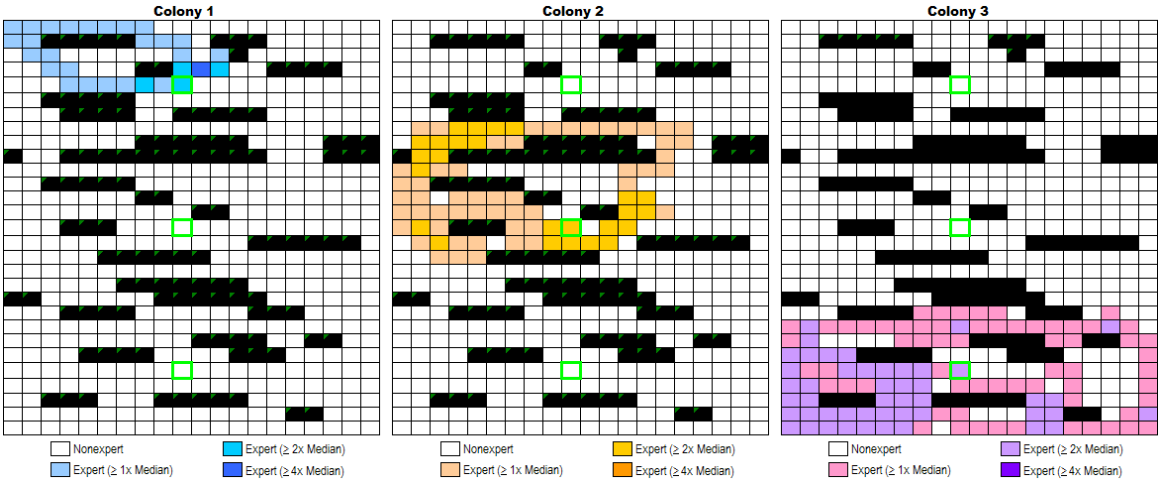

Figure 4.5: A Typical Example of Areas of Expertise as Determined through Self-Assessment (After 5 Time Steps).

visual standpoint. While there are technically several notable differences between the two, most can be explained by the fact that the results shown in the latter are not run to convergence; in fact, by running ACS-GRIDWORLD for only five time steps (shown in Figure 4.5), the resulting AOE is remarkably similar to those reported in Figure 4.4. Based on these observations, it is reasonable to assume that the ACS-GRIDWORLD's behavior over 5 time steps is comparable to that of the robots used in [1]. This suggests that the AOE process will yield similar results in both algorithm domains.

4.3.2 Determining the Expertness of Others. Whereas self-determinations of AOE are trivial, the assessment of other colonies' expertness is an inherently more difficult task [1]. In Figure 4.6, the extracted AOE of the entire gridworld (using the same data from Figure 4.3) is presented from each colonies' perspective. Because colonies do not assess the expertise of others in locations where they already possess expert knowledge, there are significant portions of the landscape which each colony ignores (shown in gray). Even so, one can clearly see that the Parzen classifier's ability to identify expertness can vary widely. In the case of colonies 2 and 3, Parzen classification appears to be highly adept at identifying expertness based solely upon pheromone values. Yet as observed in colony 1, this approach is not full-proof, and can result in colonies mistakenly believing that there are regions of the gridworld (denoted in white) where no colonies possess expert information when they actually do. The above findings are not surprising, since the results of Table 4.6 indicate that Parzen classifiers are somewhat unreliable at identifying expert states. As the above data indicates through colonies 2 and 3, though, the technique sometimes works well.

For the purposes of comparison, the extraction of others' AOEs as reported in [1] is presented in Figure 4.7. A cursory glance between these results and those presented in this section show that the two only vaguely resemble one another since the latter is run to convergence while the former is not. Conceptually, though, both figures demonstrate the same behavior; the only difference is that the overlapping of AOEs

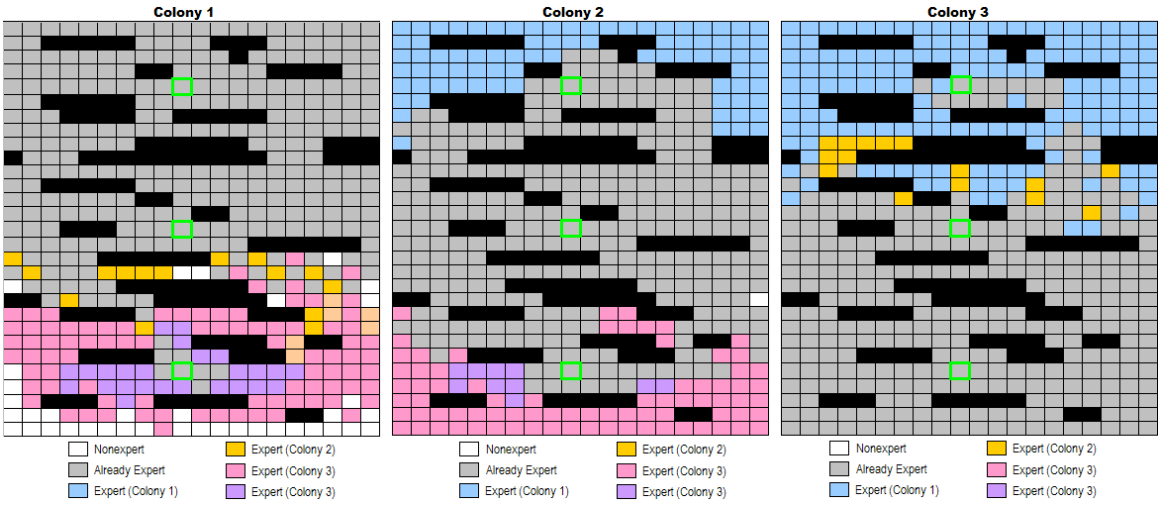

Figure 4.6: Determination of Others' Expertness from Figure 4.3 using Parzen Classification.

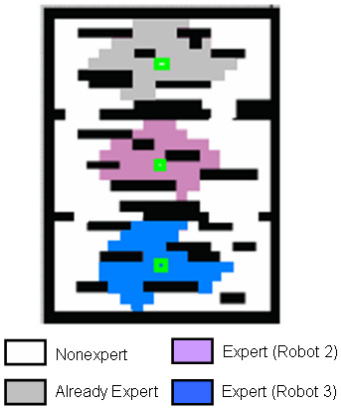

Figure 4.7: Determination of Others' Expertness from the Perspective of a Single Robot as Reported in [1].

in the ACS-GRIDWORLD example (caused by ACS' larger exploration of the search space) prevents the extraction of AOEs from being appearing as straightforward as it is in [1]. Thus for all intents and purposes, these results are considered to be similar, and serve as further proof that the use of AOE learning in ACO is indeed possible.

4.3.3 Sharing Expert Information. The final step of AOE learning calls for the sharing of expert knowledge between colonies. As can be seen in Figure 4.10, each colony augments its pheromone tables by incorporating the expert knowledge contained by others (as determined by the Parzen classifier) for each nonexpert state; in the event that no expert information is detected for a particular location, an average of all three colonies' pheromone values at that cell is used instead. The results of this synchronization step are immediately noticeable, as each colony in Figure 4.10 contains expert information in over 94% of the landscape, resulting in a gain of at least 28% for each colony. Moreover, because colonies are required to keep their expert information, another outcome of the AOE learning process is that each colony possesses a slightly different combination of expert knowledge/pheromone values. This outcome is overlooked in [1], but could be important in parallel ACO implementations since it encourages a more diverse exploration of the search space.

Although pheromone trails only influence the behavior of ants in ACO, these matrices can also be thought of as representing a colony's overall decision "policy" over the search space. In Figure 4.8, the pheromone matrices of colonies 1, 2 and 3, respectively, are shown before and after the synchronization process. In some cases, such as with colony 2, the resulting pheromone matrix is only slightly augmented by AOE learning. Other times, the difference in pheromones before and after synchronization is more noticeable, as is the case with colonies 1 and 3. For each case, however, the use of synchronization appears to be beneficial, as the pheromone policies created through AOE sharing more closely resemble the optimal policy for this problem (Figure 4.9, as determined by the Bellman Backup Equation [28]). In fact, according to Figure 4.11, synchronization improves the quality of each policy

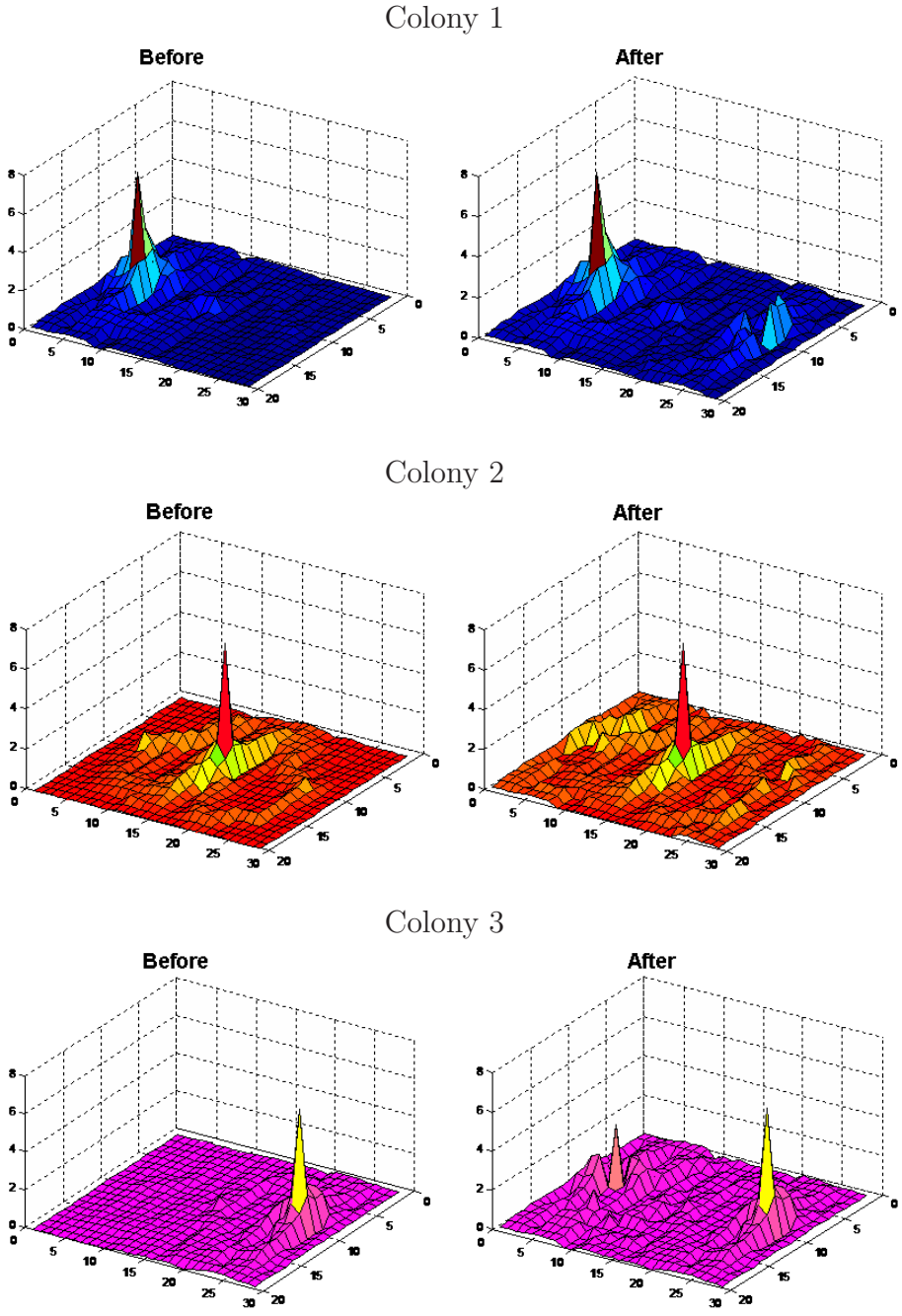

Figure 4.8: Pheromone Concentrations of Colonies 1, 2, and 3 Before and After AOE Learning.

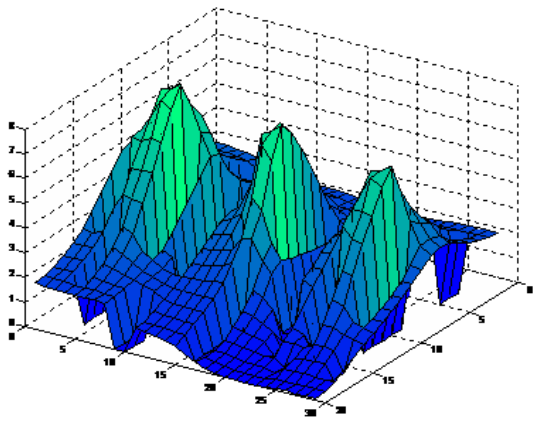

Figure 4.9: Optimal Policy for the Imanipour Benchmark as Determined by the Bellman Backup Equation [28].

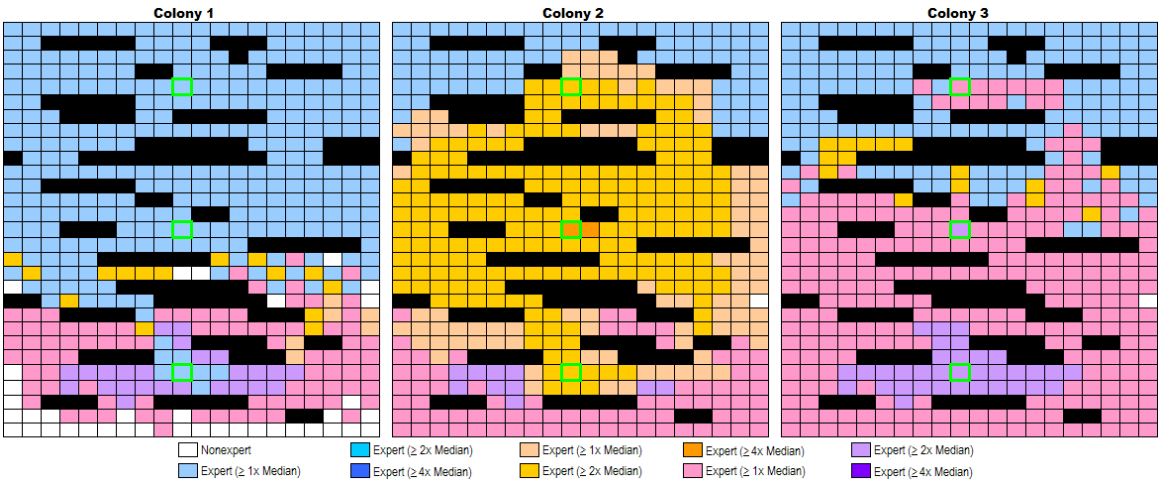

Figure 4.10: AOEs of Colonies After Synchronization.

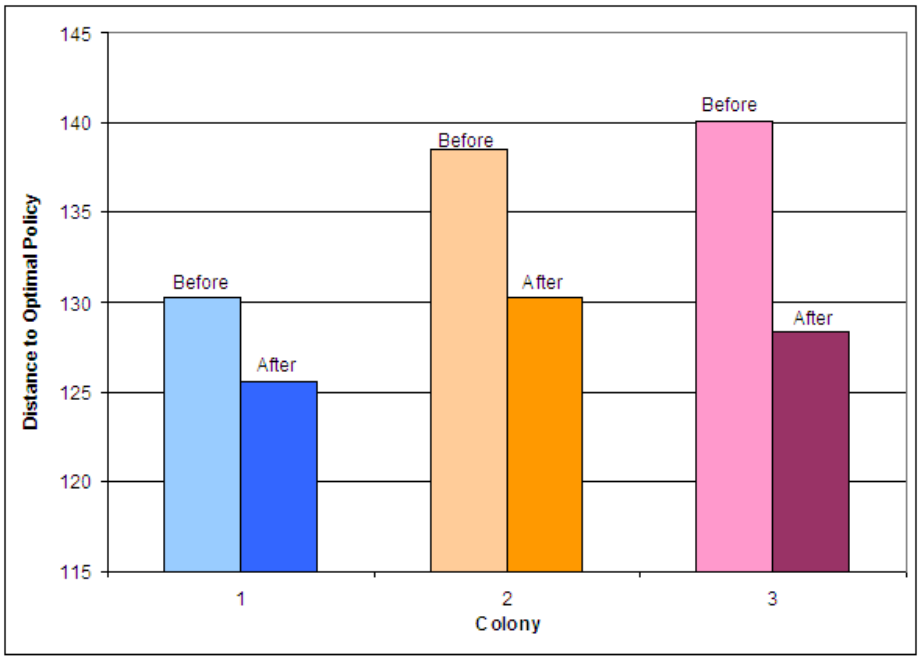

Figure 4.11: Comparison of Policies Before and After Synchronization (Lower Distance is Better).

Table 4.8: Runtime Performance of Parallel vs. Standalone ACS-GRIDWORLD (After 25 Time Steps).

	# of CPUs	# of Ants	Average Runtime (Seconds)
Parallel ACS-GRIDWORLD (with AOE)	3	3	2.869 ± 0.575
ACS-GRIDWORLD	1	3	5.54 ± 0.283

by 5.97%, on average. Thus, it is clear that the AOE learning process is doing more good than harm.

4.3.4 Performance. A comprehensive evaluation of AOE's performance within ACS-GRIDWORLD must be considered with regards to both computational time and solution quality. Thus, for the purpose of this section, the parallel ACS-GRIDWORLD implementation discussed above is compared to the performance of a single large colony. Both algorithms are run for 30 trials using the same number of time steps (25) and same number of ants (3). The only difference between the two, is the fact that the former utilizes AOE learning across multiple colonies while the latter does not.

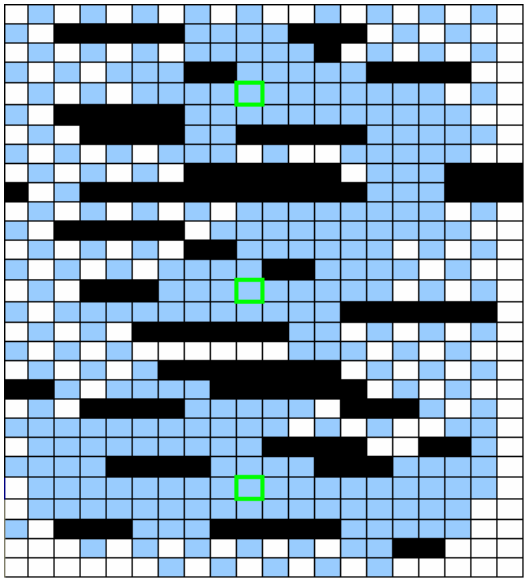

Figure 4.12: Area of Expertise of a Single ACS-GRIDWORLD Colony (After 25 Time Steps).

In terms of runtime performance, the results of Table 4.8 indicate that the use of AOE learning is preferable over the use of a single large colony. Theoretically, utilizing three processing units should allow for a speedup of exactly 3.0 over a standalone colony, since each can be delegated one-third of the entire workload. In actuality, though, the additional computational overhead imposed by the AOE process from network communication delay and classifier training reduces this speedup to approximately 1.93.

Unfortunately, while the use of AOE learning is advantageous in terms of speedup, the same cannot be said with regards to solution quality. Although an examination of AOE in Figure 4.12 indicates that ACS-GRIDWORLD develops up to 30% fewer expert states when run as a single large colony, the resulting pheromone matrix (Figure 4.13) of this single colony more closely approximates the optimal policy than any of the matrices produced through AOE learning (Figure 4.14). At first glance, this discovery discredits the notion of using AOE learning in ACO, as it suggests that the increases in runtime performance afforded by this technique are more than offset by significant losses in solution quality. Closer analysis, though, reveals

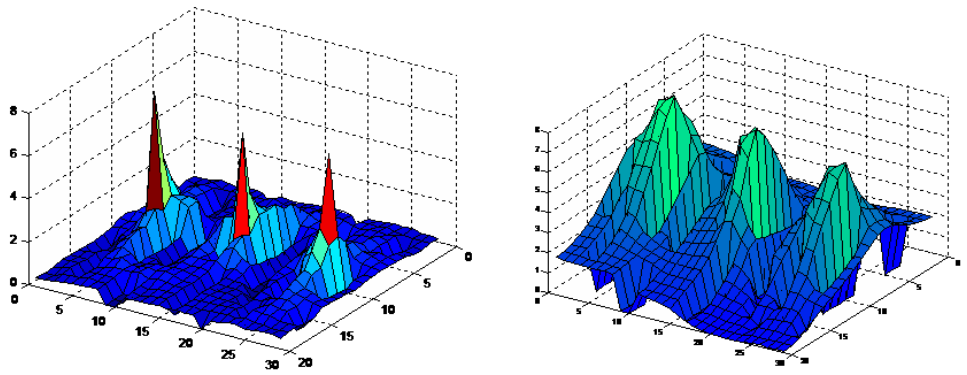

Figure 4.13: Pheromone Concentrations of a Single ACS-GRIDWORLD Colony After 25 Time Steps (Left) Compared to an Optimal Policy (Right).

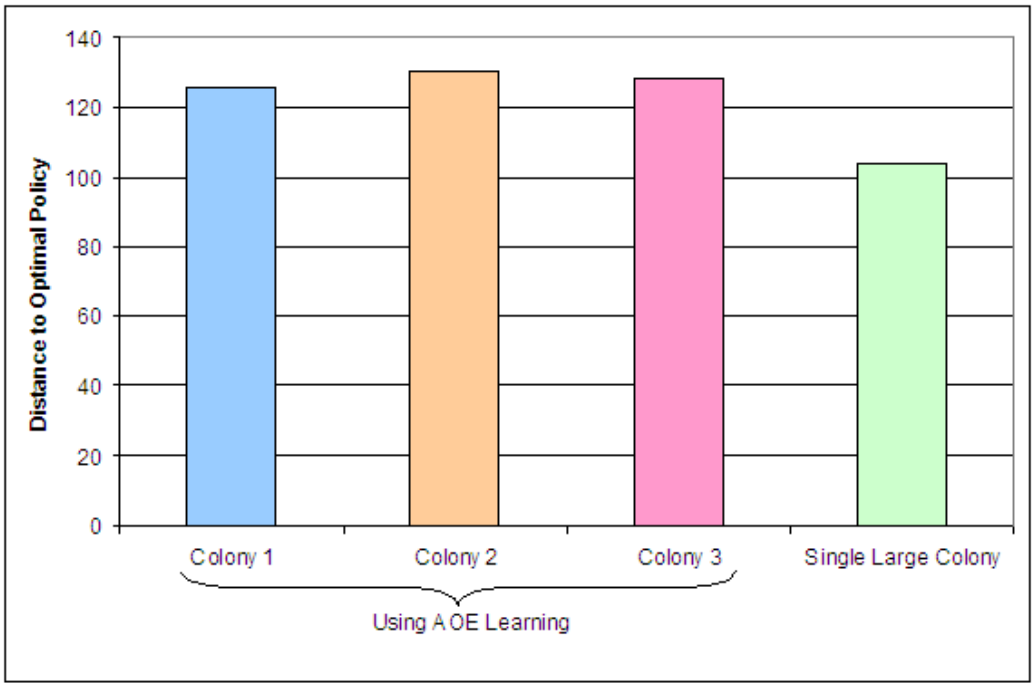

Figure 4.14: Comparison of Policies Formed through ACO with AOE Learning versus a Single Large Colony (Lower Distance is Better).

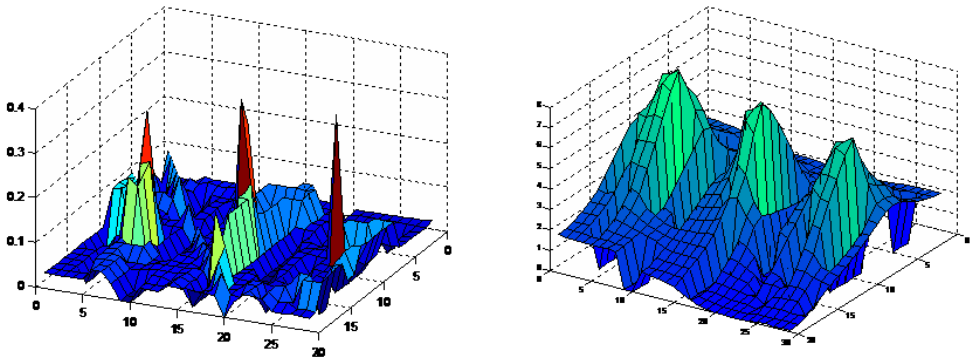

Figure 4.15: Combined Pheromone Policy of Three ACS-GRIDWORLD Colonies Synchronized Just Before Overlapping of AOE Occurs (Left) Compared with Optimal Policy (Right).

that these findings are the result of a fundamental shortcoming of the AOE strategy, as discussed next.

As previously mentioned, all of the results reported in this section are generated by running ACS-GRIDWORLD until convergence. While this approach maximizes the amount of expert information in which each colony can share, an unexpected side effect is that it can also cause each colony to learn *too much* about the landscape. As indicated in Figure 4.8, each ACS-GRIDWORLD colony learns the path to the nearest goal exceptionally well. The problem, though, is that by the time each colony achieves convergence, it also considers itself expert at the other two goal locations (Figure 4.3). Thus, while sharing pheromones between colonies should result in a combined policy similar to the one shown in Figure 4.13, AOE's self-imposed restriction of not overwriting expert knowledge prevents each colony from learning the pheromone "peaks" at all three goals. In light of these findings, it becomes apparent that effectiveness of AOE learning, regardless of the problem domain, largely depends upon the time at which synchronization occurs. As demonstrated through this section, waiting too long can cause colonies to refuse useful expert knowledge. If synchronization occurs too soon, though, both the quality and quantity of expert information which can be shared is significantly reduced. In order to demonstrate this principle, three ACS-GRIDWORLD colonies were allowed to run until their respective

AOEs touched each other (determined by combining each's AOE at the end of every time step). Synchronizing at that exact time step (which varied for each run but was found on average to occur at the 7th time step) results in the pheromone policy shown in Figure 4.15. This pheromone matrix more closely resembles an optimal policy than any of the other AOE policies created in this section and was created in a third of the time, thus proving that AOE learning can be highly effective when used properly.

4.3.5 Closing Remarks. Based on these results, the use of AOE learning in ACO is a mixed bag. On the one hand, sharing expert knowledge does appear to enhance the quality of policies generated by each participating colony in a manner similar to that reported in [1]. In order to maximize the effectiveness of this exchange, however, synchronization must occur within a narrow timeframe where each colony has enough expert information to share but not enough to become set in its ways. Nevertheless, the findings obtained in this section serve as proof that the AOE learning technique can be successfully applied to the ACO framework. As a result, the following sections focuses on examining the use of this technique when directly applied to a more traditional ant colony problem domain.

4.4 Impact of ACO with AOE Learning in the TSP

With the use of AOE learning in ACO validated through gridworld, the focus of this thesis now turns towards the use of ACO combined with AOE learning with regards to the TSP. Using the parallelized version of ACS-TSP developed in Chapter III, this section evaluates the effectiveness of AOE learning when solving various TSPLIB [26] benchmarks using networks of 2 and 5 colonies. Performance is measured by comparing the solutions generated through this method against both those created by a single large colony as well as through parallel independent runs, which currently represent the best known means of using ACO according to [15] and [23], respectively.

Since the effectiveness of AOE learning is reliant upon exchanging large quantities of meaningful information, the issue of when to synchronize is also closely con-

Table 4.9: Effects of Single Versus Multiple Time Step Synchronization on the Ulysses22 Benchmark (Optimal Answer is 75.67).

2 Processors			
Synch Time Step	Best	Worst	μ
Only at 10	76.79	81.88	79.511 ± 1.056
Only at 25	76.03	81.06	78.953 ± 1.182
Only at 50	76.74	81.57	79.29 ± 1.005
Only at 100	76.10	81.04	78.858 ± 1.153
Only at 200	76.22	81.79	79.221 ± 1.2
Every 10	76.44	81.53	79.14 ± 1.036
Every 25	76.28	81.23	78.778 ± 1.166
Every 50	76.47	81.33	78.985 ± 0.958
Every 100	76.12	81.23	79.301 ± 1.167
Every 200	76.12	82.20	79.243 ± 1.073

5 Processors			
Synch Time Step	Best	Worst	μ
Only at 10	76.03	81.82	77.427 ± 1.357
Only at 25	76.03	82.09	77.559 ± 1.375
Only at 50	76.03	81.78	77.513 ± 1.261
Only at 100	76.03	82.80	77.618 ± 1.385
Only at 200	76.03	81.93	77.504 ± 1.298
Every 10	76.03	81.36	77.458 ± 1.072
Every 25	76.09	81.45	77.616 ± 1.283
Every 50	76.09	81.72	77.519 ± 1.244
Every 100	76.03	82.33	77.437 ± 1.256
Every 200	79.03	82.02	77.527 ± 1.298

sidered in this section. Throughout the following experiments, a variety of synchronization schedules ranging from one-time exchanges to swapping after every few time steps are implemented and analyzed. Due to the results of the previous section, however, only those schedules which swap at least once before convergence (defined in the TSP as the point where ants no longer find a better solution) are considered. For the TSP problems examined, convergence typically occurs no later than after the 200th time step. The results produced by these varying synchronization schedules are then compared to one another in the following section in hope that doing so will reveal an optimal synchronization strategy.

4.4.1 Single Versus Multiple Time Step Synchronization. Because it is only meant to demonstrate proof-of-concept, the use of AOE learning in ACS-GRIDWORLD is considered in terms of a one-time exchange of information between colonies. It is conceivable, however, that a more effective means of utilizing AOE learning is to share expert information multiple times during a single trial. This section examines the impact of various synchronization schedules on ACS-TSP which occur at either a fixed time step or after an interval of time steps. The following experiments utilize the ulyssess22, att48, and st70 benchmarks, and consist of 30 trials. For multiple synchronization experiments, colonies swap in multiples of 10, 25, 50, 100 or 200 time steps. For single synchronization experiments, colonies swap at only the 10th, 25th, 50th, 100th, or 200th time step. A summary of the solutions generated from each experiment is reported in Tables 4.9, 4.10, and 4.11.

Table 4.10: Effects of Single Versus Multiple Time Step Synchronization on the ATT48 Benchmark (Optimal Answer is 33523.71).

	2 Processors			5 Processors			
Synch Time Step	Best	Worst	μ	Synch Time Step	Best	Worst	μ
Only at 10	35709.80	37514.75	36727.979 ± 410.365	Only at 10	34966.17	38313.68	36702.635 ± 588.755
Only at 25	34880.18	37584.69	36769.661 ± 441.122	Only at 25	35336.56	37832.40	36704.229 ± 524.313
Only at 50	35153.60	37874.34	36704.929 ± 503.504	Only at 50	35090.71	37933.19	36705.532 ± 576.92
Only at 100	35469.84	37947.10	36777.356 ± 460.555	Only at 100	35236.79	38103.44	36719.374 ± 563.957
Only at 200	35520.57	37551.66	36788.722 ± 478.725	Only at 200	35399.47	37668.88	36733.781 ± 497.096
Every 10	35838.26	37784.24	36835.439 ± 428.561	Every 10	35250.39	38040.40	36752.496 ± 581.399
Every 25	35610.98	37790.00	36734.563 ± 524.927	Every 25	35046.42	38181.10	36688.165 ± 589.128
Every 50	35194.74	38176.73	36628.356 ± 567.441	Every 50	35391.41	37822.91	36691.108 ± 508.748
Every 100	35215.15	37738.07	36802.487 ± 533.966	Every 100	34698.45	37900.50	36726.05 ± 585.523
Every 200	35705.55	38009.28	36838.397 ± 497.391	Every 200	34875.25	38102.06	36823.662 ± 541.753

Table 4.11: Effects of Single Versus Multiple Time Step Synchronization on the ST70 Benchmark (Optimal Answer is 678.59).

	2 Processors			5 Processors			
Synch Time Step	Best	Worst	μ	Synch Time Step	Best	Worst	μ
Only at 10	751.59	784.48	766.216 ± 7.376	Only at 10	740.77	805.70	776.748 ± 9.384
Only at 25	747.22	786.73	768.635 ± 7.782	Only at 25	743.52	801.87	776.902 ± 10.949
Only at 50	747.22	871.52	767.163 ± 8.545	Only at 50	744.41	800.70	775.773 ± 10.423
Only at 100	749.20	785.30	767.571 ± 6.84	Only at 100	746.93	804.48	777.644 ± 9.21
Only at 200	754.19	784.27	768.729 ± 6.144	Only at 200	741.43	795.60	777.054 ± 9.517
Every 10	749.39	486.58	769.355 ± 8.839	Every 10	750.19	802.68	778.493 ± 9.767
Every 25	753.26	785.53	770.439 ± 7.009	Every 25	740.25	798.50	778.137 ± 9.756
Every 50	744.26	785.98	768.103 ± 7.596	Every 50	738.35	802.61	778.157 ± 10.064
Every 100	751.20	788.91	769.858 ± 7.546	Every 100	748.68	797.66	777.82 ± 9.345
Every 200	737.50	783.34	769.253 ± 8.117	Every 200	750.10	796.79	778.533 ± 8.94

In terms of solution quality, an inspection of these tables indicates that there is no single synchronization schedule that consistently stands out with regards to either best or average case performance. In fact, it is even difficult to determine a synchronization strategy that generally works best since ulyssess22 benefits the most from a single synchronization at time step 25 (or several other schedules), st70 profits from multiple exchanges every 50 or 200 time steps (depending upon whether or not 2 or 5 colonies are being utilized, respectively), and att48 benefits from either synchronizing just once at the 25th time step with 2 colonies or after every 100 time steps when using 5. However, the results of a 5% t-test indicates that the differences in solution quality between single and multiple step synchronization policies are statistically negligible across all three benchmarks. As a result, keeping the number of colonies constant, these results suggest that either approach is a valid means of synchronizing knowledge in the TSP.

Although there is no discernable difference in terms of solution quality, there is a noticeable disparity between single and multiple step synchronization schedules with regards to runtime performance. As mentioned in Section 4.2.1, synchronization is a nontrivial process, since training a Parzen classifier and sharing information causes colonies to expend effort that could be used in exploring/exploiting the search space. Consequently, synchronization schedules that use a single synchronization step always run significantly faster than those that that repeatedly share information. In light of these findings, the motivation for utilizing multiple synchronization steps is severely reduced since the use of a single synchronization schedule can produce similar results in a fraction of the time. For this reason, the following sections in this chapter forgo the use of multiple synchronization steps in favor of a single swap approach.

4.4.2 Performance. Unlike in ACS-GRIDWORLD, where the concept of AOE learning can easily be expressed and evaluated visually, the complexity of the TSP prevents a similar visualization from being as informative. As a result, the performance of parallel ACO with AOE learning in this problem domain is more effectively examined in terms of solution quality and run times. In Tables 4.12 through 4.16, the effectiveness of ACS-TSP using AOE learning is compared to that of a single large colony as well as to parallel independent runs [23]. For the purposes of this section, the number of benchmarks examined is expanded to include the eil51 and berlin52 TSPLIB problems. Moreover, the synchronization schedules used for each AOE experiment borrows heavily from the lessons learned in the previous section. As a result, synchronization occurs only once per episode in order to maximize efficiency, and the number of time steps before swapping is focused on a narrower range (5, 10, 15, 20, 25, 30, 35, 40, 45, 50); in Tables 4.10, 4.11, and 4.9 synchronizing once between the 5th and 50th time steps resulted in the discovery of the highest quality solution.

With regards to solution quality, the results of Tables 4.12 through 4.16 provide several useful insights as to the effectiveness of AOE learning when applied to ACS-TSP. In terms of best case performance, AOE learning appears to improve the

Table 4.12: Comparison of Solution Quality for ACS-TSP using a Single Large Colony, Parallel Independent Runs, or AOE Learning on the Ulysses22 Benchmark (Optimal Answer is 75.67).

	2 Processors			5 Processors			
Synch Time Step	Best	Worst	μ	Synch Time Step	Best	Worst	μ
Single Large Colony	78.03	78.95	78.333 ± 0.19	Single Large Colony	78.03	78.95	78.333 ± 0.19
Parallel Independent Runs	77.62	80.89	78.449 ± 0.53	Parallel Independent Runs	76.10	79.38	78.105 ± 0.477
5	78.03	79.25	78.373 ± 0.405	5	76.10	79.21	78.015 ± 0.54
10	77.95	80.89	78.395 ± 0.546	10	76.10	78.39	78.021 ± 0.455
15	77.07	79.25	78.357 ± 0.394	15	76.10	79.16	78.067 ± 0.427
20	77.62	80.89	78.531 ± 0.566	20	76.44	78.39	78.126 ± 0.289
25	77.62	80.24	78.328 ± 0.417	25	76.10	79.08	78.119 ± 0.348
30	77.07	79.25	78.435 ± 0.458	30	76.03	78.39	78.078 ± 0.397
35	77.95	79.25	78.325 ± 0.301	35	76.10	79.21	78.081 ± 0.461
40	77.95	80.24	78.418 ± 0.474	40	76.10	79.38	77.99 ± 0.629
45	77.07	79.25	78.429 ± 0.459	45	76.10	79.38	78.075 ± 0.522
50	77.95	81.33	78.445 ± 0.604	50	76.03	79.38	78.094 ± 0.471

Table 4.13: Comparison of Solution Quality for ACS-TSP using a Single Large Colony, Parallel Independent Runs, or AOE Learning on the ATT48 Benchmark (Optimal Answer is 33523.71).

	2 Processors			5 Processors			
Synch Time Step	Best	Worst	μ	Synch Time Step	Best	Worst	μ
Single Large Colony	36014.89	37222.19	36672.702 ± 334.76	Single Large Colony	36014.89	37222.19	36672.702 ± 334.76
Parallel Independent Runs	35222.76	37246.04	36637.894 ± 401.563	Parallel Independent Runs	34881.42	37411.80	36133.028 ± 529.049
5	35503.94	37401.57	36727.497 ± 404.694	5	34612.62	37730.96	36214.635 ± 541.449
10	35899.44	37442.38	36813.266 ± 326.866	10	34538.91	37654.33	36233.842 ± 529.372
15	35490.05	37419.17	36681.044 ± 433.347	15	34517.78	37765.46	36098.326 ± 531
20	35651.16	37453.09	36715.179 ± 366.234	20	34517.78	37678.39	36210.423 ± 599.598
25	35662.56	37419.44	36633.177 ± 427.968	25	34875.25	37626.10	36242.061 ± 550.175
30	35428.54	37306.63	36686.149 ± 387.154	30	34665.37	37562.98	36246.942 ± 606.388
35	35779.30	37418.72	36684.933 ± 341.884	35	34561.69	37653.20	36174.675 ± 578.498
40	35541.30	37510.46	36653.169 ± 422.679	40	34640.31	37441.62	36141.433 ± 530.105
45	35490.05	37381.80	36568.579 ± 436.836	45	34344.10	37153.42	36136.639 ± 525.623
50	35590.67	37501.78	36704.936 ± 383.341	50	34554.39	37941.20	36266.989 ± 618.617

Table 4.14: Comparison of Solution Quality for ACS-TSP using a Single Large Colony, Parallel Independent Runs, or AOE Learning on the Eil51 Benchmark (Optimal Answer is 426).

	2 Processors			5 Processors			
Synch Time Step	Best	Worst	μ	Synch Time Step	Best	Worst	μ
Single Large Colony	462.55	474.45	469.52 ± 3.32	Single Large Colony	462.55	474.45	469.52 ± 3.32
Parallel Independent Runs	456.45	473.95	468.344 ± 5.429	Parallel Independent Runs	454.13	479.11	467.122 ± 5.34
5	456.45	474.32	467.662 ± 4.851	5	451.04	482.71	467.089 ± 5.664
10	460.92	474.45	469.25 ± 4.068	10	453.30	484.19	466.914 ± 5.693
15	456.45	476.79	468.078 ± 4.957	15	453.36	486.03	467.451 ± 5.599
20	456.45	474.32	468.263 ± 4.967	20	441.16	486.31	466.926 ± 6.242
25	459.52	473.12	468.148 ± 4.352	25	453.36	481.91	466.395 ± 5.597
30	456.45	476.65	469.06 ± 4.815	30	455.78	484.07	467.168 ± 5.234
35	460.19	473.95	468.58 ± 4.194	35	455.70	479.52	466.672 ± 5.107
40	456.45	477.70	468.64 ± 5.284	40	455.29	484.11	468.394 ± 5.681
45	456.45	477.70	467.678 ± 5.32	45	452.36	491.50	467.622 ± 6.447
50	456.45	473.12	468.12 ± 4.922	50	450.05	481.18	466.971 ± 5.287

Table 4.15: Comparison of Solution Quality for ACS-TSP using a Single Large Colony, Parallel Independent Runs, or AOE Learning on the Berlin52 Benchmark (Optimal Answer is 7542)

	2 Processors			5 Processors			
Synch Time Step	**Best**	**Worst**	**μ**	**Synch Time Step**	**Best**	**Worst**	**μ**
Single Large Colony	7708.71	7881.02	7738.987 ± 56.42	Single Large Colony	7708.71	7881.02	7738.987 ± 56.42
Parallel Independent Runs	7708.71	8009.69	7774.466 ± 83.784	Parallel Independent Runs	7704.92	8091.57	7874.65 ± 80.759
5	7708.71	8006.52	7784.217 ± 84.302	5	7708.71	8051.51	7861.873 ± 85.752
10	7708.71	7962.28	7775.211 ± 75.722	10	7704.92	8106.35	7854.075 ± 93.011
15	7708.71	7930.06	7761.021 ± 71.178	15	7708.71	8118.49	7857.221 ± 93.655
20	7708.71	7946.73	7776.351 ± 77.107	20	7708.71	8150.67	7880.718 ± 91.65
25	7708.71	8007.66	7779.178 ± 83.481	25	7704.92	8101.73	7862.15 ± 93.997
30	7704.08	7961.76	7768.886 ± 72.232	30	7708.71	8026.15	7852.413 ± 88.03
35	7708.71	7906.57	7777.956 ± 72.747	35	7704.92	8050.46	7860.489 ± 79.077
40	7708.71	7962.28	7773.071 ± 76.805	40	7704.92	8146.43	7871.883 ± 91.904
45	7708.71	8025.31	7793.33 ± 83.571	45	7708.71	8097.29	7869.172 ± 91.855
50	7704.08	7998.78	7776.607 ± 82.808	50	7704.92	8075.33	7865.273 ± 93.393

Table 4.16: Comparison of Solution Quality for ACS-TSP using a Single Large Colony, Parallel Independent Runs, or AOE Learning on the ST70 Benchmark (Optimal Answer is 426)

	2 Processors			5 Processors			
Synch Time Step	**Best**	**Worst**	**μ**	**Synch Time Step**	**Best**	**Worst**	**μ**
Single Large Colony	747.22	765.86	758.994 ± 5.41	Single Large Colony	747.22	765.86	758.994 ± 5.41
Parallel Independent Runs	747.22	775.52	760.184 ± 6.142	Parallel Independent Runs	752.90	794.59	774.086 ± 9.138
5	742.77	775.77	761.725 ± 7.693	5	745.98	792.60	771.632 ± 10.461
10	747.22	775.42	761.76 ± 6.291	10	743.81	790.41	772.018 ± 10.296
15	747.22	776.25	761.415 ± 7.013	15	737.50	791.60	773.516 ± 9.35
20	745.91	774.21	761.846 ± 7.772	20	743.81	793.18	772.51 ± 9.066
25	742.00	773.57	760.492 ± 6.705	25	743.81	793.12	773.533 ± 9.66
30	742.00	778.16	759.705 ± 7.795	30	749.88	790.52	774.585 ± 9.214
35	745.29	776.22	761.055 ± 7.326	35	743.81	793.86	772.739 ± 9.928
40	746.18	773.75	760.672 ± 6.533	40	732.51	792.81	772.716 ± 10.517
45	746.92	771.05	760.372 ± 6.398	45	743.81	794.81	773.18 ± 9.985
50	747.22	776.97	761.512 ± 6.579	50	737.47	794.87	772.593 ± 10.141

Table 4.17: Runtime Performance of a Single Large Colony, Parallel Independent Runs, and AOE Learning on the Ulysses22 Benchmark.

2 Colonies	Average Runtime (Seconds)	Speedup Over Single Large Colony	Speedup Over Area of Expertise Learning	Number of Ants/Colony
Single Large Colony	2.96 ± 0.056	-	-	22
Area of Expertise Learning (All Time Steps)	2.472 ± 0.477	1.197	-	11
Parallel Independent Runs	2.575 ± 0.173	1.149	0.96	11

5 Colonies	Average Runtime (Seconds)	Speedup Over Single Large Colony	Speedup Over Area of Expertise Learning	Number of Ants/Colony
Single Large Colony	2.96 ± 0.056	-	-	22
Area of Expertise Learning (All Time Steps)	1.747 ± 0.105	1.694	-	≈ 5
Parallel Independent Runs	0.859 ± 0.072	3.444	2.033	≈ 5

performance of ACS-TSP, as the best solution to all five benchmarks are discovered by ACS-TSP when utilizing this technique. An examination of average case behavior, however, provides a more realistic perspective. Although there are times when utilizing AOE learning can result in statistically better solutions than those obtained by a single large colony (for example, Table 4.13 for 5 processors), there are also instances where the technique actually hampers performance, as seen in both sides of Table 4.16. Furthermore, using five processors/colonies appears to have a discernible impact on solution quality over just two in most cases. Problems such as berlin52 and st70 reverse this trend, though as the average tour length found by five colonies is noticeably worse in these benchmarks than those generated using just two. Without a doubt, the most important finding of Tables 4.12 through 4.16 is that the solutions generated through the use of AOE learning are statistically indistinguishable from those obtained via parallel independent runs. The importance of this discovery cannot be overstated, as it indicates that the AOE technique produces tour lengths that are on part with parallel independent runs - the best known means of parallelizing ACO according to [23]. Such a result is to be expected, though, since the ACS-TSP with AOE algorithm used throughout this section is essentially parallel independent runs combined with a single synchronization step.

Table 4.18: Runtime Performance of a Single Large Colony, Parallel Independent Runs, and AOE Learning on the ATT48 Benchmark.

2 Colonies	Average Runtime (Seconds)	Speedup Over Single Large Colony	Speedup Over Area of Expertise Learning	Number of Ants/Colony
Single Large Colony	33.273 ± 0.109	-	-	48
Area of Expertise Learning (All Time Steps)	28.529 ± 0.096	1.166	-	24
Parallel Independent Runs	27.92 ± 0.214	1.192	1.022	24

5 Colonies	Average Runtime (Seconds)	Speedup Over Single Large Colony	Speedup Over Area of Expertise Learning	Number of Ants/Colony
Single Large Colony	33.273 ± 0.109	-	-	48
Area of Expertise Learning (All Time Steps)	17.007 ± 2.373	1.956	-	≈ 9
Parallel Independent Runs	7.535 ± 0.062	4.416	2.258	≈ 9

Table 4.19: Runtime Performance of a Single Large Colony, Parallel Independent Runs, and AOE Learning on the Eil51 Benchmark.

2 Colonies	Average Runtime (Seconds)	Speedup Over Single Large Colony	Speedup Over Area of Expertise Learning	Number of Ants/Colony
Single Large Colony	43.611 ± 1.99	-	-	51
Area of Expertise Learning (All Time Steps)	34.385 ± 0.379	1.268	-	≈ 25
Parallel Independent Runs	33.442 ± 0.115	1.304	1.028	≈ 25

5 Colonies	Average Runtime (Seconds)	Speedup Over Single Large Colony	Speedup Over Area of Expertise Learning	Number of Ants/Colony
Single Large Colony	43.611 ± 1.99	-	-	51
Area of Expertise Learning (All Time Steps)	23.363 ± 0.651	1.867	-	≈ 10
Parallel Independent Runs	9.363 ± 0.15	4.658	2.495	≈ 10

Table 4.20: Runtime Performance of a Single Large Colony, Parallel Independent Runs, and AOE Learning on the Berlin52 Benchmark.

2 Colonies	Average Runtime (Seconds)	Speedup Over Single Large Colony	Speedup Over Area of Expertise Learning	Number of Ants/Colony
Single Large Colony	43.17 ± 0.278	-	-	52
Area of Expertise Learning (All Time Steps)	37.035 ± 0.274	1.166	-	26
Parallel Independent Runs	35.379 ± 0.119	1.22	1.046	26

5 Colonies	Average Runtime (Seconds)	Speedup Over Single Large Colony	Speedup Over Area of Expertise Learning	Number of Ants/Colony
Single Large Colony	43.17 ± 0.278	-	-	52
Area of Expertise Learning (All Time Steps)	25.438 ± 0.232	1.697	-	≈ 10
Parallel Independent Runs	9.722 ± 0.049	4.44	2.616	≈ 10

Table 4.21: Runtime Performance of a Single Large Colony, Parallel Independent Runs, and AOE Learning on the ST70 Benchmark.

2 Colonies	Average Runtime (Seconds)	Speedup Over Single Large Colony	Speedup Over Area of Expertise Learning	Number of Ants/Colony
Single Large Colony	126.881 ± 0.515	-	-	70
Area of Expertise Learning (All Time Steps)	111.096 ± 0.997	1.142	-	35
Parallel Independent Runs	106.256 ± 0.825	1.194	1.046	35

5 Colonies	Average Runtime (Seconds)	Speedup Over Single Large Colony	Speedup Over Area of Expertise Learning	Number of Ants/Colony
Single Large Colony	126.881 ± 0.515	-	-	70
Area of Expertise Learning (All Time Steps)	73.871 ± 6.268	1.718	-	14
Parallel Independent Runs	27.885 ± 0.252	4.55	2.648	14

While the use of AOE learning in ACS-TSP is validated from the perspective of solution quality, an analysis of runtime performance is less encouraging. According to Tables 4.17 through 4.21, the use of multiple small colonies is to the benefit of both AOE learning and parallel independent runs, as it allows for the same amount of work to be accomplished in a shorter amount of time. What is interesting, though, is that this speedup is not directly proportional to the number of colonies which are utilized. Even in the case of parallel independent runs, which do not require any synchronization, the need to communicate even the most basic bookkeeping information between colonies (i.e., solutions, initial pheromone values, etc.) results in a noticeable runtime penalty. Consequently, it is not surprising to see that when significant quantities of data are being exchanged, as is the case of in AOE learning, the resulting performance hits can be severe. When the number of processors/colonies is limited to just two, there appears to be little difference between AOE learning and parallel independent runs. In fact, on the ulysses22 benchmark, the use of AOE learning can actually be shown to be faster than parallel independent runs on average. Once the number of processors is increased to five, however, the shortcomings of AOE learning become evident. Because more colonies equates to more knowledge that requires synchronization, the AOE process incurs a significant runtime penalty that constantly increases with the amount of parallelization use. Note that AOE with five colonies still allows for a speedup to 1.96 over that of a single large colony (Figure 4.18). Compared to the 4.42 speedup achieved by parallel independent runs on the same problem, however, it is clear that AOE learning is at a distinct disadvantage.

Although the AOE learning technique shows some promise in the TSP, its slower runtime performance combined with virtually little change in solution quality means this technique has little if any appeal. Certainly, it may be possible to increase runtime speeds by eliminating the master/worker hierarchy (described in Chapter III) in favor of an approach that requires each colony to be responsible for performing synchronization for itself. Based on the results obtained thus far, however, the use

Table 4.22: Performance of a Single Large Colony, Parallel Independent Runs, and AOE Learning on the Eil101 Benchmark.

Solution Quality (Optimum Answer is 629)

	Average Path Length	Significantly Different to Single Large Colony? (According to 5% t-test)	Significantly Different to Parallel Independent Runs? (According to 5% t-test)
Single Large Colony	734.975 ± 11.348	-	Yes; Better
Area of Expertise Learning (After 30 Time Steps)	750.263 ± 9.429	Yes; Worse	Yes; Better
Parallel Independent Runs	757.692 ± 10.372	Yes; Worse	-

Runtime Performance

	Average Runtime (Seconds)	Speedup Over Single Large Colony	Speedup Over Area of Expertise Learning
Single Large Colony	829.785 ± 9.188	-	-
Area of Expertise Learning (After 30 Time Steps)	313.993 ± 42.829	2.643	-
Parallel Independent Runs	184.244 ± 21.905	4.504	1.704

of parallel independent runs is still the preferred means of parallelizing ACO for the TSP.

4.4.3 Application of AOE to Large TSP Problems. Although the results in this section are focused on relatively small benchmarks, the conclusions drawn from them are also applicable to larger problem instances. In order to demonstrate this, the use of AOE learning on three larger TSPLIB benchmarks (eil101, pr264, pcb442) is provided in Tables 4.22 through 4.24. As is typical of ACO, the quality of solutions generated by all three approaches decreases significantly as the size of the problem increases. However, the relationships in performance between all three techniques remain largely the same. The results of Tables 4.22 through 4.24 show that the use of a single large colony produces the highest quality results. When compared to the runtime performance of this method, however, it is clear that the slight decrease in

Table 4.23: Performance of a Single Large Colony, Parallel Independent Runs, and AOE Learning on the PR264 Benchmark.

Solution Quality (Optimum Answer is 49135)

	Average Path Length	Significantly Different to Single Large Colony? (According to 5% t-test)	Significantly Different to Parallel Independent Runs? (According to 5% t-test)
Single Large Colony	60057.518 ± 166.974	-	Yes; Better
Area of Expertise Learning (After 30 Time Steps)	61200.527 ± 555.363	Yes; Worse	No
Parallel Independent Runs	61363.408 ± 514.338	Yes; Worse	-

Runtime Performance

	Average Runtime (Seconds)	Speedup Over Single Large Colony	Speedup Over Area of Expertise Learning
Single Large Colony	31687.684 ± 41.88	-	-
Area of Expertise Learning (After 30 Time Steps)	12366.231 ± 19.83	2.562	-
Parallel Independent Runs	7884.747 ± 129.183	4.019	1.569

Table 4.24: Performance of a Single Large Colony, Parallel Independent Runs, and AOE Learning on the PCB442 Benchmark.

Solution Quality (Optimum Answer is 50778)

	Average Path Length	Significantly Different to Single Large Colony? (According to 5% t-test)	Significantly Different to Parallel Independent Runs? (According to 5% t-test)
Single Large Colony	71109.956 ± 799.336	-	Yes; Better
Area of Expertise Learning (After 30 Time Steps)	73191.363 ± 568.9	Yes; Worse	Yes; Better
Parallel Independent Runs	74362.553 ± 639.016	Yes; Worse	-

Runtime Performance

	Average Runtime (Seconds)	Speedup Over Single Large Colony	Speedup Over Area of Expertise Learning
Single Large Colony	244404.444 ± 234.098	-	-
Area of Expertise Learning (After 30 Time Steps)	97233.798 ± 8261.285	2.514	-
Parallel Independent Runs	57676.249 ± 5227.062	4.238	1.686

solution quality caused by either AOE learning or parallel independent runs is more than made up for by their respective speedups. Furthermore, while there are times when the AOE process appears to generate higher quality solutions than parallel independent runs, the fact that these gains are never more than 0.5% better makes the additional runtime costs of this method unappealing when compared to the latter, especially on larger sized TSP instances. Certainly, this examination is not completely redundant, as it demonstrates that the use of AOE learning has the potential to improve the performance of parallel ACO. Until an optimal synchronization schedule is determined, however, the method still loses out to parallel independent runs on the TSP.

4.5 Determining when to Synchronize

Throughout this chapter, the effectiveness of AOE learning in ACO has been shown to be highly dependent upon when colonies exchange expert information with one another. In gridworld, the presence of multiple known goals makes determining the optimal conditions for synchronization a simple matter of ensuring that colonies share expert knowledge before their AOEs overlap. With the TSP, however, determining the best conditions for synchronization is complicated by the fact that 1) there is only one true goal in the environment, 2) ant colonies rarely if ever reach this goal on their own, 3) ant colonies do not know if they have achieved convergence until many time steps afterwards, and 4) every colony has easy access to the same portions of the search space. As a result, utilizing a gridworld-like synchronization strategy is infeasible, as it would require examining AOEs over long stretches of time and applying synchronization retroactively. Using the data obtained from all trials of ACS-TSP conducted in the previous two sections, a variety of approaches have been studied in an attempt to isolate trends which may reveal the most effective time to share expertness in the TSP. This section contains a discussion of each of these approaches and their results.

Fixed Time Step All of the synchronization schedules experimented with in this chapter are selected based on the assumption that the best time to swap AOEs occurs after some fixed number of time steps have elapsed. As illustrated in Tables 4.13 through 4.16, the best solutions discovered with 2 colonies were found when synchronization occurs at 30 time steps. An analysis of the 5 colony runs, however, reveals that the optimal time step for synchronization is far more erratic. Based on these findings, it is assumed that the use of 30 time steps is useful as a guideline, but not as a definite rule.

Median Value in Visit Table Because AOE learning uses the median of the visit table as the threshold for expertness, it stands to reason that an optimal time to synchronize might be once the median reaches a specific value. In all of the benchmark problems tested, however, the median of the visit table is always equal to one up to the 50th time step. This is because the sheer number of edges in a TSP prevents ants from being able to visit each one prior to the 50th time step; this causes AOE to calculate the median as being one. As a result, visit table values are not believed to have any bearing on the synchronization process.

Ratio of Expert to Nonexpert States Since AOE learning places heavy emphasis on the concept of expertness, one approach to synchronization is to track the ratio of expert to nonexpert states. Doing so produces a curve like the one seen in Figure 4.16. Although it was hoped that the instantaneous slope of this curve could be used to predict the most opportune time to swap, an examination of these ratios across all eight benchmarks did not reveal a meaningful pattern.

Pheromone Concentrations Since pheromone values are a direct measure of expertness, the maximum and minimum pheromone trails within a pheromone table are tracked at each time step. A problem with this approach, however, is that the amount of pheromones deposited in ACS-TSP are a function of the path length, which causes different problems to result in different pheromone

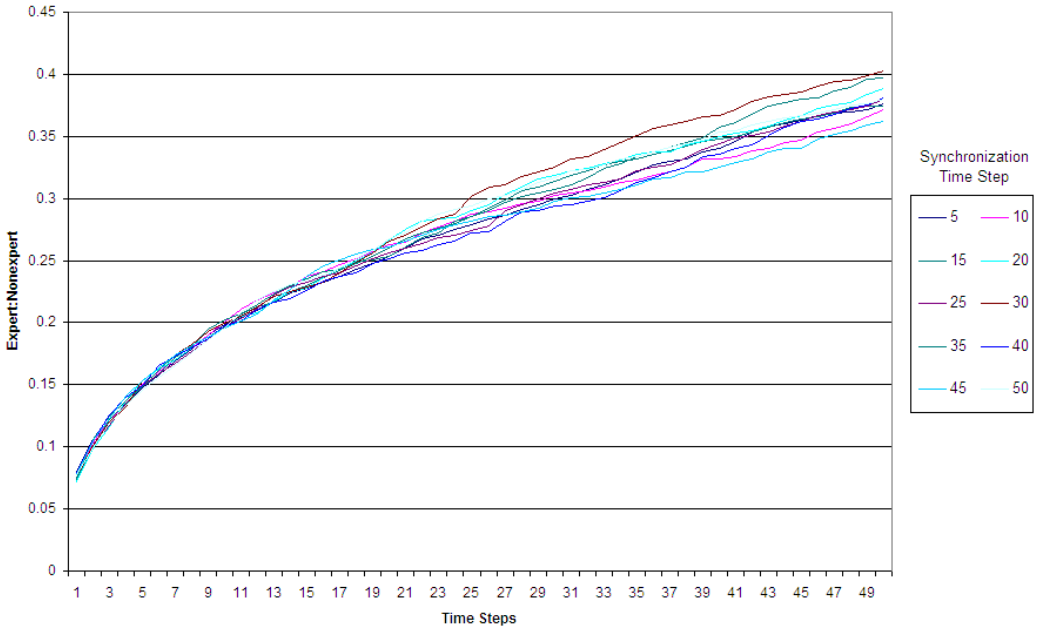

Figure 4.16: Ratio of Expert to Nonexpert States for the Berlin52 Benchmark During the First 50 Time Steps.

ranges. As a result, a direct comparison of pheromone concentrations does not produce meaningful results.

Number of Time Steps Since Max Pheromone Value has Changed In order to compensate for varying pheromone concentration ranges between benchmarks, each colony tracks the number of time steps since the maximum pheromone value has changed. This approach is based on the idea that the best time to synchronize is after the maximum pheromone value has reached a plateau. An examination of the performance of ACS-TSP on the att48 benchmark, however, reveals that the best time to synchronize occurs while the maximum pheromone value is steadily increasing. As a result, no correlation between the maximum pheromone value, the final solution, and the synchronization time step can be made.

Number of Time Steps Since Last Solution Improvement As demonstrated in Section 4.3.3, synchronizing after convergence can hurt the effectiveness of the AOE process. In an attempt to predict convergence and synchronize before it

is too late, each colony kept track of the number of time steps that had elapsed since a higher quality solution had been found. In all of the benchmarks, though, by the time convergence could be predicted with a high degree of certainty, it was already long past the optimal time to synchronize. This indicates that waiting for convergence is not an effective means of determining when to synchronize.

Number of Time Steps Since Last Significant Solution Improvement This final approach is similar to the one above, with the only difference being that convergence is considered once the colony does not make a significant i.e. 5% improvement for some time. Unfortunately, within the time window examined (5 to 50 time steps), the ant colony is constantly making significant improvements in its solution. As a result, this strategy was not found to be useful.

Despite the wide breadth of approaches considered, there does not appear to be a set of conditions which can be used to identify the best time to synchronize ACS-TSP with absolute precision. Instead, the results of this analysis indicate that while some features may appear to be indicative of an optimal synchronization schedule, they do not serve as an indicator of behavior across all instances. In many ways, the stochastic nature of ACO makes the task of predicting the effectiveness of AOE learning extremely difficult. For some cases, such as with eil51, the synchronization time step which produces the best solution also happens to produce the worst. In other cases, such as berlin52, varying the number of time steps before synchronization has practically no impact on solution quality. Based on the results of this study, it is difficult to say for certain whether or not a set of optimal synchronization conditions can be found. However, this investigation suggest that if these conditions do exist, they are not as intuitive as one might initially believe.

4.5.1 Closing Remarks. Throughout this chapter, the use of AOE learning is evaluated as a means of synchronizing the efforts of multiple ant colonies working in tandem. Although the AOE method is imperfect due to its reliance on Parzen classification to identify expertness, the approach is found to be effective at augmenting

a colony's pheromone matrix with the expert knowledge of others. Unfortunately, the impact of AOE learning on solution quality can widely vary. In multi-objective environments such as gridworld, the sharing of AOEs between ant colonies can be shown to allow for the creation of policies that are similar in quality to those generated by a single large colony, but generated in nearly half the time. In a single objective environment such as TSP, however, the use of AOE learning is less impressive, as the technique has no discernable impact on solution quality and results in slower runtime performance than parallel independent runs. Based on the results from both problem domains, it is apparent that the impact of AOE on performance largely varies depending upon when information is exchanged. Yet while an effective synchronization schedule can be determined for the gridworld, a set of similar synchronization conditions for the TSP could not be found. Further study in this area is obviously required.

V. Conclusion

This thesis has explored the application of Area of Expertise (AOE) learning within parallel Ant Colony Optimization (ACO). Specifically, it provides a means of incorporating AOE learning within the ACO framework, and analyzes the resulting algorithms when used in a multi-objective gridworld environment as well as the traveling salesman problem (TSP). In order to maximize the performance of the AOE approach, an in-depth study was conducted to identify the most opportune time to exchange expert knowledge between colonies. The approach was then compared against the best known means of parallelizing ACO, which as of this writing means parallel independent runs, as well as forgoing parallelism altogether in favor of a single large colony.

5.1 Contributions and Achievements

The major accomplishments and achievements of this thesis include the following:

- Both the traveling salesman and gridworld problems are discussed, with special emphasis placed on identifying the practical impediments (i.e. computational complexity) that make finding high quality solutions in either problem domain difficult.
- The ACO metaheuristic, an effective and efficient means of solving discrete combinatorial optimization problems, is reviewed. Using the ACS-TSP algorithm [15] as a template, a new ant colony algorithm known as ACS-GRIDWORLD is created. This algorithm represents the first known application of ACO to the gridworld environment.
- The major problems associated with parallelizing ACO are examined. A review of the current literature as provided in Chapter II indicates that while a variety of approaches have been suggested throughout the years in order to achieve cooperative learning in ACO, all have resulted in either decreased solution quality, slower runtime performance, or both.

- Area of Expertise learning [1], a novel means of sharing information between cooperative agents, is reviewed. This technique focuses on individual agents determining where they possess expert knowledge. Then, through the use of a Parzen classifier, each agent evaluates the expertness of others and augments its knowledge base whenever the former contains expert information and the latter does nota. Although AOE learning has only been applied in robotics, the principles of the technique are easily transferrable to the ACO metaheuristic. Consequently, a means for upgrading ACS-TSP and ACS-GRIDWORLD to include the AOE synchronization mechanism is provided.
- An analysis of the AOE process within parallel ACO yields the following observations concerning the performance of Parzen classification:
  - An imbalance in the accuracies between expert ($\approx$60%) and nonexpert ($\approx$90%) states indicates that Parzen classification is extremely pessimistic when determining expertness. This is due to the small number of data points that are available to train the classifier, as well as the fact that the majority of a colony's knowledge base consists of nonexpert states. As a result, the classifier often mistakes expertness for nonexpertness. Consequently, during the synchronization process, it is possible for an agent to "miss out" on a piece of expert knowledge because it is mistakenly classified as nonexpert.
  - The optimal Parzen window size does not appear to be heavily influenced by either the particular problem being solved or the number of time steps that have elapsed prior to training. Instead, the initial pheromone value τ_0 is the biggest contributor to the classifier's final dimensions. Since this value remains constant in ACS-GRIDWORLD and ACS-TSP, respectively, there is no need to search for a good window size once it has been empirically determined, thus saving significant time during classifier training.

- As proof-of-concept, the AOE learning strategy is first considered with regards to ACS-GRIDWORLD. Some of the highlights of this investigation include:
  - The behavior of AOE within ACO is found to be comparable to that reported in [1] using robots. These results indicate that the AOE technique can be successfully translated into an ant colony framework.
  - Exchanges of expert pheromone information (coinciding with one's AOE) between colonies is found to be advantageous. An evaluation of the pheromone policies created before and after synchronization indicate that the latter more closely resembles an optimum policy. This shows that synchronization can be used to improve solution quality while shortening execution time.
  - A comparison of the policies created through AOE learning and by a single large colony illustrates that the use of the former is not guaranteed to yield higher performance. If synchronization occurs too soon or too late, the amount of expert information that is exchanged is too small to be meaningful, resulting in a policy that is less optimal than one created through a single large colony.
  - The best time to share expert information is right before each colony's expertise overlaps one another. This simple synchronization criteria is only possible in multi-objective environments due to the fact that each colony can focus on a separate goal.
  - In terms of runtime performance, the use of AOE learning is found to have a significant advantage over a single large colony. Although the maximum speedup recorded is 1.93, it is expected to increase as the size of the landscape and number of ants used increases.
- Based on the success of the AOE learning when applied to gridworld, a similar configuration was applied in the single objective traveling salesman problem, and tested using the ulysses22, att48, eil51, berlin52, st70, eil101, pr264, and

pcb442 benchmarks (provided by TSPLIB). While it was hoped that utilizing AOE in combination with ACO would lead to improved performance, the results of this study instead indicated the following:

- In terms of solution quality, synchronizing after every few time steps is shown to be equivalent to synchronizing only once per episode. This, in conjunction with the computationally expensive nature of the swapping process, makes a single swap synchronization schedule the preferred means of utilizing AOE learning in ACO.
- When compared to parallel independent runs and a single large colony using the same number of ants, the use of AOE learning does not produce any significant gains in solution quality. On the contrary, the tours generated by AOE learning are nearly identical to those generated via parallel independent runs, and are slightly worse (on larger problems) than those found by a single large colony.
- With regards to runtime performance, AOE learning is at a disadvantage. Although a maximum speedup of 1.96 was obtained using AOE when compared to a single large colony, the extra computational time needed for synchronization makes it significantly slower than parallel independent runs, which achieves a speedup of 4.42 on the same problem. On average, parallel independent runs are capable of executing in half the time.
- On TSP benchmarks consisting of more than 100 nodes, the solutions generated by AOE learning is statistically better than parallel independent runs. This indicates that the use of the technique can improve solution quality, although this improvement was never observed to be greater than 0.5%.
- As was the case in gridworld, determining when to synchronize plays a key role in maximizing the usefulness of the AOE technique. Unfortunately, the single objective nature of the TSP makes the task of determining this

optimal moment difficult. Although a variety of approaches were considered, none were found to be useful in identifying the precise moment when synchronizing expert knowledge is most advantageous.

5.2 Future Work

The research conducted in this thesis demonstrates that AOE learning can be beneficial in either single or multiple objective environments. At the same time though, it also illustrates that the method is prone to inaccurate classifications of expertness, slower runtime performance, limited improvements in solution quality, and synchronization timing issues. In this regard, there is clearly a need for further work and testing. The main goal of any future research should be focused on improving the quality of expert information that is shared between colonies, as well as improving the speed at which this information is shared. The following list provides some ideas for future work:

- Removal of Parzen classification in favor of a less computationally expensive process for determining expertness. One possible candidate for this is to simply use each colony's visit table threshold.
- Replacement of the master/worker hierarchy in favor of a more decentralized topology. In this thesis, the master was found to be a performance bottleneck during the synchronization process. By replacing it with a peer-to-peer network, the task of synchronization can then be delegated to the colonies themselves. This should greatly speed up the speed at which colonies are able to share information with one another.
- Further evaluation of the impact of AOE learning on even larger sized TSP instances. This should determine if the marginal solution improvements observed in Chapter IV on larger problems is a trend or a one-time occurrence.
- Perform additional experiments solely designed to uncover the optimal synchronization conditions in a single objective environment. Based on this thesis, it

is known that pheromone concentrations, visit table values, and solution convergence does not appear to be the sole indicator of when to synchronization. It may be the case, however, that determining when to synchronize depends on several conditions.

- Incorporate AOE learning into other ACO systems such as Win or Learn Fast (WoLF) Ant [7]in order to determine if the technique has any discernable impact.
- Develop a means of "rewinding" an ACO episode such that different synchronization schedules can be tested on the same problem. This allows for the impact of AOE learning to be precisely measured, instead of trying to approximate its behavior through multiple trials.
- Create a visualization tool that shows the AOE of each colony in real time. Such a tool should allow researchers to better determine the optimal conditions for synchronization in both single and multi-objective environments.
- Finally, future work should include the application of AOE learning in other problems for which ACO has already been applied. This includes, but is not limited to the Quadratic Assignment Problem, Multidimensional Knapsack Problem, Vehicle Routing Problem, and other various scheduling problems.

Bibliography

1. Ahmadabadi, Majid Nili, Ahmad Imanipour, Babak N. Araabi, Masoud Asadpour, and Roland Siegwart. "Knowledge-based Extraction of Area of Expertise for Cooperation in Learning". *Proc. of the IEEE/RSJ Conf. on Intelligent Robots and Systems [IROS]*. Beijing, China, Oct. 2006.

2. Applegate, David L., Robert E. Bixby, Vasek Chvatal, and William Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 41 William Street, Princeton, New Jersey, USA, 08540-5237, 2007.

3. Arabshahi, Payman, Andrew Gray, I. Kassabalidis, M.A. El-Sharkawi, R.J. Marks II, A. Das, and S. Narayanan. "Adaptive Routing in Wireless Communication Networks using Swarm Intelligence". *9th AIAA Int. Communications Satellite Systems Conf., 17-20 April 2001, Toulouse, France*. 2001. URL `citeseer.ist.psu.edu/arabshahi01adaptive.html`.

4. Becker, T.J. "No Accidental Tourist: Solving the Traveling Salesman Problem Has Been a 16-year Passion for Researcher — Research Horizons Magazine", 2004. URL `http://gtresearchnews.gatech.edu/newsrelease/salesman.htm`. [Online; accessed 02-May-2007].

5. Bentley, Jon Louis. "Experiments on traveling salesman heuristics". *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, 91–99. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1990. ISBN 0-89871-251-3.

6. Bonabeau, E., M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, NY, first edition, 1999.

7. Bowling, Michael H. and Manuela M. Veloso. "Multiagent learning using a variable learning rate". *Artificial Intelligence*, 136(2):215–250, 2002. URL `citeseer.ist.psu.edu/bowling02multiagent.html`.

8. Bullnheimer, B., G. Kotsis, and C. Strauss. "Parallelization Strategies for the Ant System", 1997. URL `citeseer.ist.psu.edu/bullnheimer98parallelization.html`.

9. Catalano, Maria Stella Fiorenzo and Federico Malucelli. "Parallel randomized heuristics for the set covering problem". 113–132, 2001.

10. Cioni, L. "Some Strategies for Parallelizing Ant Systems", 2005. URL `http://www.di.unipi.it/ lcioni/papers/2005/ArtAnts.pdf`.

11. Comellas, F. and J. Ozon. "An ant algorithm for the graph colouring problem", 1998. URL `citeseer.ist.psu.edu/comellas98ant.html`.

12. Dawande, M., J. Kalagnanam, P. Keskinocak, R. Ravi, and F. S. Salman. "Approximation Algorithms for the Multiple Knapsack Problem with Assignment Restrictions". *Combinatorial Optimization*, 4(2):171–186, 2000.

13. Delisle, P., M. Krakecki, M. Gravel., and C. Gagne. "Parallel Implementation of an Ant Colony Optimization Metaheuristic with OpenMP", 2001.

14. Donati, A., L. Gambardella, N. Casagrande, A. Rizzoli, and R. Montemanni. "Time Dependent Vehicle Routing Problem with an Ant Colony System". URL `citeseer.ist.psu.edu/562261.html`.

15. Dorigo, Marco and Luca Maria Gambardella. "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem". *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, April 1997. URL `citeseer.ist.psu.edu/article/dorigo96ant.html`.

16. Dorigo, Marco, Vittorio Maniezzo, and Alberto Colorni. "The Ant System: Optimization by a colony of cooperating agents". *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41, 1996. URL `citeseer.ist.psu.edu/dorigo96ant.html`.

17. Ellabib, Issmail, Paul Calamai, and Otman Basir. "Exchange strategies for multiple Ant Colony System". *Inf. Sci.*, 177(5):1248–1264, 2007. ISSN 0020-0255.

18. F. Kruger, M. Middendorf, D. Merkle. "Studies on a Parallel Ant System for the BSP Model", 1998.

19. Gambardella, Luca Maria and Marco Dorigo. "Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem". *International Conference on Machine Learning*, 252–260. 1995. URL `citeseer.ist.psu.edu/33860.html`.

20. Kruger, F., D. Merkle, and M. Middendorf. "Studies on a Parallel Ant System for the BSP Model", 1998.

21. Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, first edition, 1985.

22. López-Ibáñez, M., L Paquete, and T. Stützle. "On the Design of ACO for the Biobjective Quadratic Assignment Problem". M. Dorigo, M. Birattari, C. Blum, L. Gambardella, F. Montada, and T. Stützle (editors), *4th International Workshop on Ant Colony Optimization (ANTS 2004)*, volume 3172 of *Lecture Notes in Computer Science*, 214–225. Springer Verlag, 2004. URL `papers/ants2004.pdf`. (©Springer Verlag).

23. Manfrin, Max, Mauro Birattari, Thomas Stutzle, and Marco Dorigo. "Parallel ACO for the Traveling Salesman Problem", 2006. URL `http://code.ulb.ac.be/dbfiles/ManBirStuDor2006ants.pdf`.

24. Michalewicz, Z. and D. Fogel. *How to Solve It: Modern Heuristics*. Springer, Verlag, Berlin, second edition, 2004.

25. Middendorf, Martin, Frank Reischle, and Hartmut Schmeck. "Information Exchange in Multi Colony Ant Algorithms". *Lecture Notes in Computer Science*, 1800:645+, 2000. URL citeseer.ist.psu.edu/article/middendorf00information.html.

26. Reinelt, Gerhard. "TSPLIB 95", 2007. URL http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.

27. Richard O. Duda, Peter E. Hart and David G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, NY, USA, 2001. ISBN 0-471-05669-3.

28. Russell, S. and P. Norvig. *Artifical Intelligence: A Modern Approach*. Pearson, Upper Saddle River, New Jersey, second edition, 2003.

29. St, T. and u Hoos. "MAX MIN Ant System", 2000. URL citeseer.ist.psu.edu/312713.html.

30. Stützle, Thomas. "Parallelization Strategies for Ant Colony Optimization". *Lecture Notes in Computer Science*, 1498:722–??, 1998. URL citeseer.ist.psu.edu/utzle98parallelization.html.

31. Taillard, E. D. and L. Gambardella. *An ant approach for structured quadratic assignment problems*. Technical Report IDSIA-22-97, 22, 1997. URL citeseer.ist.psu.edu/taillard97ant.html.

32. Talbi, Roux, Fonlupt, and Robillard. "Parallel Ant Colonies for Combinatorial Optimization Problems". *Feitelson & Rudolph (Eds.), Job Scheduling Strategies for Parallel Processing: IPPS '95 Workshop, Springer LNCS 949*, volume 11. 1999. URL citeseer.ist.psu.edu/talbi99parallel.html.

33. Tsai, Cheng-Fa, Chun-Wei Tsai, and Ching-Chang Tseng. "A new hybrid heuristic approach for solving large traveling salesman problem". *Inf. Sci. Inf. Comput. Sci.*, 166(1-4):67–81, 2004. ISSN 0020-0255.

34. Tschoeke, S., R. Luling, and B. Monien. "Solving the traveling salesman problem with a distributed branchand -bound algorithm on a 1024 processor network", 1995. URL citeseer.ist.psu.edu/148117.html.

35. van der Zwaan, S. and C. Marques. "Ant Colony Optimisation for Job Shop Scheduling", 1999. URL citeseer.ist.psu.edu/vanderzwaan99ant.html.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
13-09-2007	Master's Thesis	Sept 2006 – Sept 2007

TITLE AND SUBTITLE

PARALLELIZATION OF ANT COLONY OPTIMIZATION VIA AREA OF EXPERTISE LEARNING

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

AUTHOR(S)

de Freitas, Adrian, 2nd Lieutenant, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/07-15

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Ant colony optimization algorithms have long been touted as providing an effective and efficient means of generating high quality solutions to NP-hard optimization problems. Unfortunately, while the structure of the algorithm is easy to parallelize, the nature and amount of communication required for parallel execution has meant that parallel implementations developed suffer from decreased solution quality, slower runtime performance, or both. This thesis explores a new strategy for ant colony parallelization that involves Area of Expertise (AOE) learning. The AOE concept is based on the idea that individual agents tend to gain knowledge of different areas of the search space when left to their own devices. After developing a sense of their own expertness on a portion of the problem domain, agents share information and incorporate knowledge from other agents without having to experience it firsthand. This thesis shows that when incorporated within parallel ACO and applied to multi-objective environments such as a gridworld, the use of AOE learning can be an effective and efficient means of coordinating the efforts of multiple ant colony agents working in tandem, resulting in increased performance.

15. SUBJECT TERMS

Ant Colony Optimization, Area of Expertise Learning, Metaheuristic Optimization

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Chris B. Mayer, Maj, USAF (ENG)
U	U	U	UU	114	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-3636, ext 4542 (Chris.Mayer@afit.edu)